I parametri e il sovraccarico

Vediamo dei dettagli sul passaggio dei parametri.

Parametri formali e attuali

```
class Esempio {
  static void primo(int x) {
    int y;
  }

  public static void main(String args[]) {
    primo(24+3);
  }
}
```

variabile locale `y`
: è una variabile che viene creata quando si invoca il metodo `prova`

parametro attuale `24+3`
: è il *valore* che viene trasmesso al metodo

parametro formale `x`
: è una variabile del metodo `primo`, il cui valore iniziale è quello del parametro attuale

I parametri formali sono come le variabili locali tranne che per il valore iniziale.

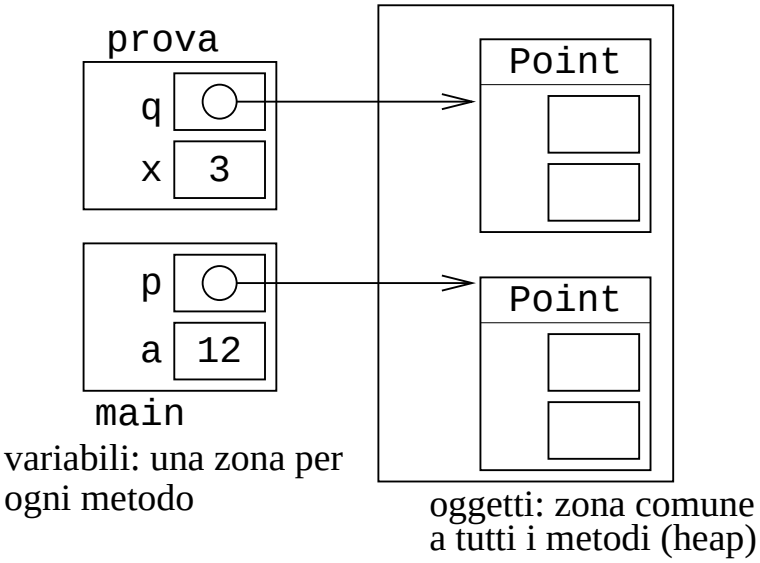
Il parametro attuale è un valore, non una variabile.

Memoria degli oggetti

Ogni metodo ha una sua zona di memoria in cui ci sono le variabili.

Queste zone *non comprendono gli oggetti*, che stanno da un'altra parte.

```
class Esempio {
  static void prova() {
    int x;
    Point q;
    q=new Point();
  }

  public static void main(String args[]) {
    int a;
    Point p;
    p=new Point();
  }
}
```



Una zona per ogni metodo, più una zona comune a tutti per gli oggetti.

Passare un oggetto a un metodo

In generale:

- passare un parametro=assegnare al parametro formale quello attuale

Nel caso in cui il parametro sia un oggetto:

- assegnare=assegnare l'indirizzo
- gli oggetti stanno in una zona accessibile a tutti i metodi

Mettendo insieme: nel parametro formale ci va a finire l'indirizzo dell'oggetto passato.

Quindi: l'oggetto si può modificare nel metodo, e questa modifica è visibile al metodo chiamante.

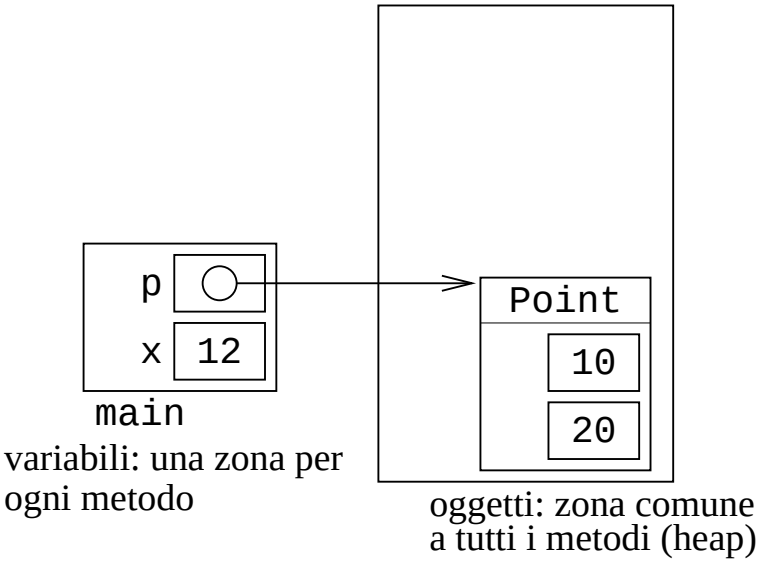
Passare un oggetto: esempio

Cosa stampa questo programma?

```
import java.awt.*;

class Modifica {
  static void azzera(int x, Point p) {
    x=0;

    p.x=0;
    p.y=0;
  }

  public static void main(String args[]) {
    int x=12;
    Point p=new Point();
    p.x=10;
    p.y=20;
```

```
    azzera(x, p);

    System.out.println(x);
    System.out.println(p);
  }
}
```

Memoria, prima di invocare il metodo



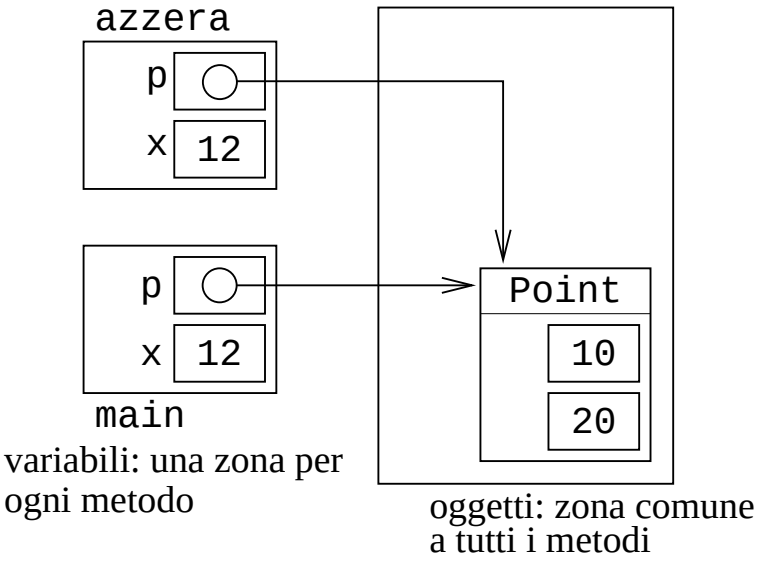
Esiste solo la zona delle variabili di `main`

La zona per `azzera` viene creata solo quando si invoca `azzera`

Memoria, quando si invoca il metodo

Viene creata la nuova zona in cui ci sono i parametri formali `x` e `p` e le variabili locali (in questo caso, nessuna)

Nei parametri formali vengono messi i valori passati.



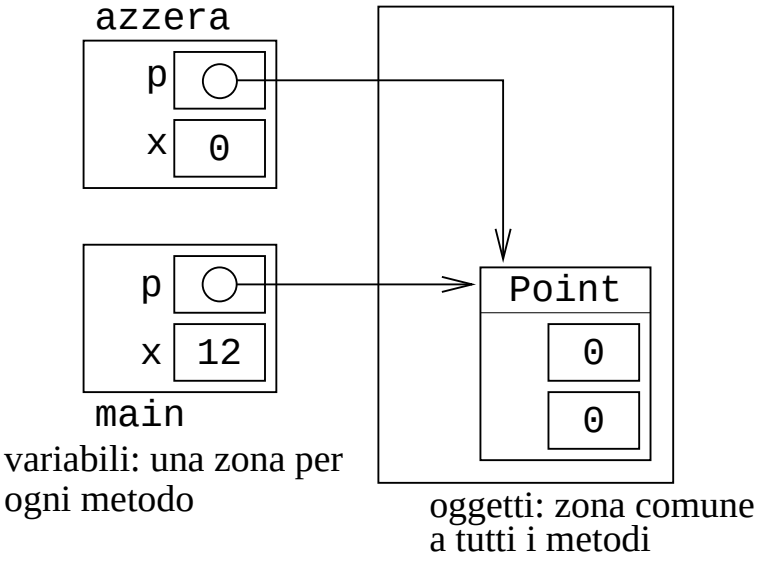
Valori passati: valore di `x`, e valore di `p`

Il valore di `p` è l'indirizzo in cui si trova l'oggetto.

L'esecuzione del metodo

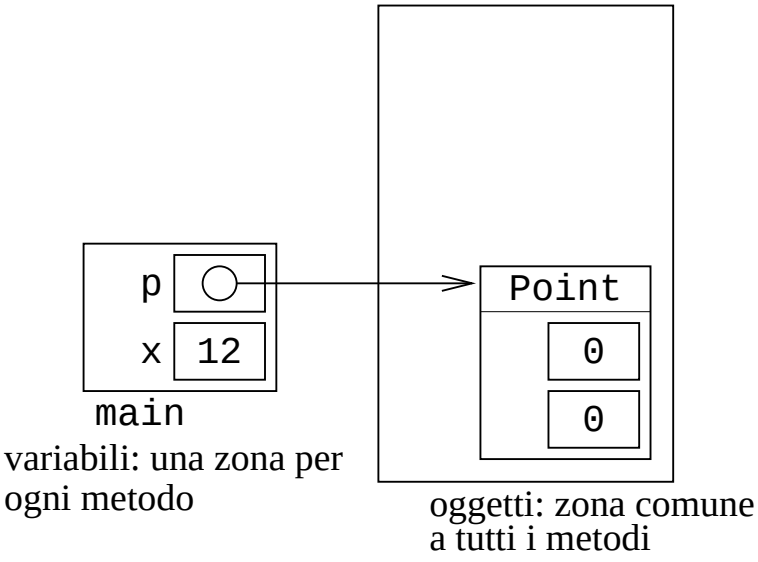
`x=0` mette `0` nella variabile locale.

`p.x=0;` e `p.y=0;` modificano l'oggetto *il cui indirizzo sta in* `p`



Cosa viene stampato?

Quando il metodo termina, viene eliminata la sua zona di memoria.



L’effetto globale

Quando passo una variabile, questa non viene modificata: `x` mantiene il valore `12`

Quando passo un oggetto, questo può venire modificato `p.x` diventa `0`

Non è questa la regola!

È una conseguenza della regola dei puntatori.

Il sovraccarico

Definire più metodi con lo stesso nome.

Serve quando la stessa operazione si può fare con diversi tipi di dato.

Regola sui parametri

Nella invocazione, i parametri devono essere del tipo usato nella dichiarazione.

```
static void sbaglio(int x) ...


sbaglio(...)
```

Fra le parentesi ci devo mettere un `int`

Regola generale

Il parametro attuale va messo nel parametro formale.

I tipi devono poter permettere questo assegnamento.

Esempio: se il parametro formale è `double`, posso passare un intero.

Se il parametro formale è intero, non posso passare un `double`, perchè l'assegnamento non si può fare.

Metodi sovraccarichi

`System.out.println()` è un metodo

Però funziona su tutti i tipi!

Intuitivamente: potrei dover fare le stesse cose su tipi diversi.

Come definire un metodo che lavora su più tipi diversi:

definire un metodo per ognuno dei tipi

Esempio: stampare le coordinate di un punto

Passo un `Point` come parametro:

```
public class UnPunto {
  static void stampaPunto(Point p) {
    System.out.println("Punto: (", p.x, ",", p.y, ")");
  }

  ...
}
```

Se ho due interi?

Voglio stampare le coordinate del punto.

Devo prima creare un oggetto punto.

```
  public static void main(String args[]) {
    Point p=new Point();
    p.x=12;
    p.y=43;

    stampaPunto(p);
  }
```

Oppure: faccio un nuovo metodo

Alternativa: creo un metodo diverso:

```
  static void stampaPuntoDueInteri (int x, int y) {
    System.out.println("Punto: (", x, ",", y, ")");
  }
```

Il sovraccarico

> *due metodi possono anche avere lo stesso nome, basta che il numero degli argomenti o il tipo di un argomento sia diverso*

Dato che i due metodi per disegnare un punto hanno argomenti diversi, posso usare lo stesso nome.

Posso usare il nome `stampaPunto` per tutti e due.

```
import java.awt.*;

public class MetPunto {
  static void stampaPunto(Point p) {
    System.out.println("Punto: (", p.x, ",", p.y, ")");
  }

  static void stampaPunto(int x, int y) {
    System.out.println("Punto: (", x, ",", y, ")");
  }

  public static void main(String args[]) {
    Point p=new Point();
    p.move(12,43);

    stampaPunto(p);

    stampaPunto(12, 32);
  }
}
```

Si può fare se il numero di argomenti è diverso.

Oppure se il tipo di almeno un argomento è diverso.

Qualche argomento può anche avere lo stesso tipo.

L’invocazione di un metodo

Il compilatore va a vedere il numero e il tipo dei parametri *attuali* (i valori che il programma manda al metodo)

Fra tutti i metodi con quel nome, sceglie quello che ha parametri di pari numero e tipo.

I valori di ritorno

I valori di ritorno dei metodi possono anche essere diversi.

```
class Prova {
  static int metodo(int x) {
    return 0;
  }

  static double metodo(double x) {
    return 0;
  }

  ...
}
```

Però...

Cosa non si può fare

I valori di ritorno possono anche essere diversi.

Però...

Non si possono definire due metodi che differiscono **solo** per il valore di ritorno.

```
  // errore!

  int metodo() { ... }
  double metodo() { ... }
```

Non fatevi ingannare dal nome

In generale: metodi sovraccarichi sono metodi diversi con lo stesso nome.

Dato che hanno lo stesso nome, per capire quale sto invocando, guardo il tipo/numero degli argomenti.

È per questo che esiste la regola che due metodi con lo stesso nome devono avere numero e/o tipo diverso di argomenti.

Esercizio

Definire un metodo che calcola la somma di due numeri.

I due numeri possono essere reali o interi (il valore di ritorno è di conseguenza)

Soluzione

Mi servono due metodi, uno per sommare interi e uno per sommare reali.

Le firme:

```
  static int somma(int, int)
  static double somma(double, double)
```

I tipi dei valori di ritorno non hanno importanza.

I tipi degli argomenti si.

Sono diversi: si possono usare i due metodi.

Soluzione

Scrivo il corpo dei due metodi.

```
class DueSomme {
  static int somma(int x, int y) {
    return x+y;
  }

  static double somma(double x, double y) {
    return x+y;
  }

  public static void main(String args[]) {
    System.out.println(somma(12,23));
    System.out.println(somma(12.3,23.1));
    System.out.println(somma(12,23.1));
  }
}
```

Nota: posso anche usare gli stessi nomi per i parametri formali (`x` e `y`) nei due metodi (ogni metodo vede solo le sue variabili).

Chi viene invocato da `somma(12,23.1)`?

Ho un intero e un reale.

Viene invocato quello dei reali.

Regola generale: viene invocato quello che ha esattamente gli stessi tipi, se c'è, altrimenti quello che li può assegnare tutti ai parametri formali.

In questo caso: non sono due interi, per cui non posso usare il metodo con i due interi.

Posso assegnare `12` e `23.1` a due reali, per cui uso il metodo con due reali.

Posso fare più di due metodi con lo stesso nome?

Si

Posso fare due metodi con lo stesso nome che fanno cose completamente diverse?

Si: sono a tutti gli effetti due metodi diversi:

```
class Diversi {
  static void aaa(int x) {
    System.out.println("Questa e' una stringa");
  }

  static void aaa(double x) {
    System.out.println(x+2);
  }

  public static void main(String args[]) {
    aaa(1);
    aaa((double) 1);
  }
}
```

Non è però consigliabile farlo.
(il programma non si capisce più)

Numero degli argomenti

Si possono fare due metodi diversi se hanno numero di argomenti diverso.

Esempio: metodo che stampa la somma di interi, al quale posso passare da zero a tre interi.

Soluzione

Devo fare quattro metodi diversi.

```
class SommaInt {
  static int somma() {
    return 0;
  }

  static int somma(int x) {
    return x;
  }

  static int somma(int x, int y) {
    return x+y;
  }

  static int somma(int x, int y, int z) {
    return x+y+z;
  }

  public static void main(String args[]) {
    System.out.println(somma(2,3));
    System.out.println(somma(2,3,4));
    System.out.println(somma());
    System.out.println(somma(2));
  }
}
```

Posso invocare questo metodo dall’altro?

Si può fare.

Se ho un oggetto, posso invocare tutti i metodi della classe (poi vedremo i modificatori di accesso).

```
class SommaStesso {
  static int somma() {
    return 0;
  }

  static int somma(int x) {
    return x;
  }

  static int somma(int x, int y) {
    return x+y;
  }

  static int somma(int x, int y, int z) {
    return somma(x,y)+z;
  }

  public static void main(String args[]) {
    System.out.println(somma(2,3));
    System.out.println(somma(2,3,4));
    System.out.println(somma());
    System.out.println(somma(2));
  }
}
```