

Programmazione 1 - Modulo C

Metodi in Java

Marco Beccuti

Università degli Studi di Torino

Dipartimento di Informatica

Ottobre 2020



Java: oggetti e classi

- È un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica;
- Un programma in Java è costituito da **oggetti** di vario tipo che interagiscono tra loro
- **oggetti** sono caratterizzati da:
 - ▶ **Campi** (Memorizzano lo stato dell'oggetto);
 - ▶ **Metodi** (Interazione con lo stato dell'oggetto e comunicazione tra oggetti).

Java: oggetti e classi

- L'oggetto è **l'istanza** di una **classe**;
- classe é quindi la definizione di un tipo di oggetto;
- Possono essere create diverse istanze della stessa classe;

```
1 public class Quadrato {  
2  
3     private int lato;  
4     /**metodo Costruttore**/  
5     public Quadrato(int lt) {  
6         lato = lt;  
7     }  
8     /**metodo per il calcolo dell'area**/  
9     public int CalcolaArea(){  
10        return lato*lato;  
11    }  
12  
13 }
```

private VS **public**

Java: private VS public

- **private**: utilizzabile solo all'interno della stessa classe;
- **public**: utilizzabile ovunque.

Java: istanziazione

```
1 public class Quadrato {  
2  
3     private int lato;  
4     /**metodo Costruttore**/  
5     public Quadrato(int lt) {  
6         lato = lt;  
7     }  
8     /**metodo per il calcolo dell'area**/  
9     public int CalcolaArea(){  
10        return lato*lato;  
11    }  
12  
13 }
```

```
1 public class CreaQuadrato {  
2     public static void main (String [] arg){  
3  
4         Quadrato q1 = new Quadrato(3);  
5         Quadrato q2 = new Quadrato(6);  
6         System.out.println("Area di q1: "+q1.CalcolaArea());  
7         System.out.println("Area di q2: "+q2.CalcolaArea());  
8  
9     }  
10 }
```

Java: metodi di istanza

- **metodi di istanza** vanno sempre invocati sull'istanza di una classe (cioè un oggetto)

```
1 public class Quadrato {
2
3     private int lato;
4     /**metodo Costruttore**/
5     public Quadrato(int lt) {
6         lato = lt;
7     }
8     /**metodo per il calcolo dell'area**/
9     public int CalcolaArea(){
10         return lato*lato;
11     }
12
13 }
```

```
1 public class CreaQuadrato {
2     public static void main (String [] arg){
3
4         Quadrato q1 = new Quadrato(3);
5         Quadrato q2 = new Quadrato(6);
6         System.out.println("Area di q1: "+q1.CalcolaArea());
7         System.out.println("Area di q2: "+q2.CalcolaArea());
8
9     }
10 }
```

Java: metodi di statici

- le classi possono offrire direttamente dei servizi tramite i **metodi statici**;
- un metodo statico va invocato sulla classe;
- l'esecuzione del metodo non prevede la presenza di un oggetto;

```
1 public class Quadrato {
2
3     private int lato;
4     /**metodo Costruttore*/
5     public Quadrato(int lt) {
6         lato = lt;
7     }
8     /**metodo per il calcolo dell'area*/
9     public int CalcolaArea(){
10        return lato*lato;
11    }
12    /**metodo statico*/
13    static String Tipo(){
14        return "Quadrato";
15    }
16 }
```

```
1 public class CreaQuadrato {
2     public static void main (String [] arg){
3
4         Quadrato q1 = new Quadrato(3);
5         Quadrato q2 = new Quadrato(6);
6         System.out.println("Area di q1: "+q1.CalcolaArea());
7         System.out.println("Area di q2: "+q2.CalcolaArea());
8         /**Uso di metodo statico*/
9         System.out.println("Tipo: "+Quadrato.Tipo());
10    }
11 }
```

Java: metodi di statici

In questo corso vedremo solo metodi statici!!!

Metodo in dettaglio

- metodo è un blocco di dichiarazioni e istruzioni con un nome ed eventuali parametri, che può essere invocato da un altro metodo per eseguire tali istruzioni;
- i metodi rendono il codice più leggibile ed compatto;
- si evita di riscrivere più volte un blocco di istruzioni.

```
1  class Esempio {  
2      static void saluto () {  
3          System.out.println("Ciao! ");  
4      }  
5  }  
6  public static void main (String[] args) {  
7      saluto ();  
8  } //fine main  
9  }  
10 } //fine classe
```

Dichiarazione di un metodo

La dichiarazione di un metodo ha la seguente sintassi:

- Modificatori: di visibilità (*private*, *public*, ...) o di appartenenza a classe o istanze (*static*);
- Tipo: del risultato restituito dal metodo;
- Nome: del metodo;
- Parametri formali: lista (anche vuota) di coppie. $\langle \text{tipo}, \text{parametro} \rangle$

```
/**  
 < modificatori > < tipo > < nome > ( < lista_parametri_formali > ) {  
    ....  
}  
**/  
  
public static int minimo ( int a , int b ) {  
    if ( a < b ) return a;  
    else return b;  
}
```

Metodi statici

Classificazione di un metodo:

- **metodi con tipo**;
- **metodi senza tipo**.

Un metodo con tipo è dichiarato nel modo seguente:

```
/**  
< modificatori > < tipo > <nome > ( < lista_parametri_formali > ) {  
    ....  
}  
**/  
  
public static int minimo ( int a , int b ) {  
    if ( a < b ) return a;  
    else return b;  
}
```

Osserva che il tipo dell'oggetto restituito da **return** deve avere lo stesso tipo dichiarato per il metodo.

Metodi statici

Classificazione di un metodo:

- **metodi con tipo**;
- **metodi senza tipo**.

Un metodo senza tipo è dichiarato nel modo seguente:

```
/**  
< modificatori > void <nome > ( < lista_parametri_formali >) {  
    ....  
}  
**/  
  
public static void stampa ( int a , int b) {  
    System.out.println("a:"+a+" b"+b);  
}
```

Osserva il comando **return** può essere usato senza farlo seguire da nessuna espressione .

Metodi statici: forme di invocazione

- Per i metodi con tipo:
se un metodo `static int m()` è dichiarato in una classe *C*, allora la sua invocazione:
 - ▶ in *C*: è un'espressione di tipo `int` della forma `m()`
 - ▶ in un'altra classe *D*: è un'espressione di tipo `int` della forma `C.m()`

`int a = SavitchIn.readLineInt();`

- Per i metodi senza tipo:
se un metodo `static void m()` è dichiarato in una classe *C*, allora la sua invocazione:
 - ▶ in *C*: è un'istruzione della forma `m()`
 - ▶ in un'altra classe *D*: è un'istruzione della della forma `C.m()`

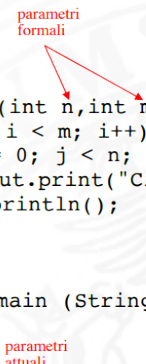
`System.out.println("CIAO");`

Invocazione di un metodo

```
class Esempio {  
    static void saluti (int n,int m) {  
        for (int i = 0; i < m; i++){  
            for (int j = 0; j < n; j++){  
                System.out.print("Ciao! ");  
                System.out.println();  
            }  
        }  
    }  
  
    public static void main (String[] args) {  
        saluti (5,3);  
    } //fine main  
} //fine classe
```

parametri formali

parametri attuali

A diagram with two red labels: 'parametri formali' at the top and 'parametri attuali' at the bottom. Two red arrows originate from 'parametri formali' and point to the parameters 'n' and 'm' in the 'saluti' method signature. Two red arrows originate from 'parametri attuali' and point to the arguments '5' and '3' in the 'saluti' method call within the 'main' method.

Esecuzione:

```
Ciao! Ciao! Ciao! Ciao! Ciao!  
Ciao! Ciao! Ciao! Ciao! Ciao!  
Ciao! Ciao! Ciao! Ciao! Ciao!
```

Passaggio dei parametri

- Il **passaggio dei parametri** è il meccanismo che lega i **parametri attuali** di una specifica invocazione di un metodo ai **parametri formali** della corrispondente dichiarazione del metodo;
- Esistono varie modalità di passaggio dei parametri: per valore, per riferimento, ...;
- Il linguaggio Java adotta il passaggio di **parametri per valore**.

Passaggio dei parametri per valore

- Se un parametro è di **tipo elementare** (es. int, boolean, double, char, ...), durante il passaggio dei parametri al parametro formale viene assegnato il valore risultante dalla valutazione del parametro attuale;
- Eventuali modifiche del parametro formale non si ripercuotono all'esterno del metodo.

```
1 public class PassaggioValore {
2
3     public static void main (String[] args) {
4         int n = 30;
5         System.out.println("n vale " + n); /** Stampa 30 **/
6         nonModifica(n);
7         System.out.println("n vale ancora " + n); /** Stampa 30 **/
8     }
9
10    public static void nonModifica(int i) {
11        System.out.println("i vale " + i); /** Stampa 30 **/
12        i = 0;
13        System.out.println("adesso i vale " + i); /** Stampa 0 **/
14    }
15 }
```


Passaggio dei parametri per valore

- Se il tipo di un parametro è un **tipo riferimento** (un array o una classe, come String, ...), durante il passaggio dei parametri al parametro formale viene assegnato un riferimento all'oggetto o array risultante dalla valutazione del parametro attuale.;
- Eventuali modifiche fatte a tale oggetto nel metodo invocato saranno visibili dal metodo chiamante dopo la fine dell'esecuzione.

```
1 public class PassaggioRiferimento {
2
3     public static void main (String[] args) {
4         int[] array = new int[1];
5         array[0] = 30;
6         System.out.println("array[0] vale " + array[0]); // Stampa 30
7         modifica(array);
8         System.out.println("adesso array[0] vale " + array[0]); // Stampa 0
9     }
10
11     public static void modifica(int[] a) {
12         System.out.println("a[0] vale " + a[0]); // Stampa 30
13         a[0] = 0;
14         System.out.println("adesso a[0] vale " + a[0]); // Stampa 0
15     }
16 }
```

Memoria nella JVM

La memoria nella JVM si può pensare sostanzialmente divisa in due parti:

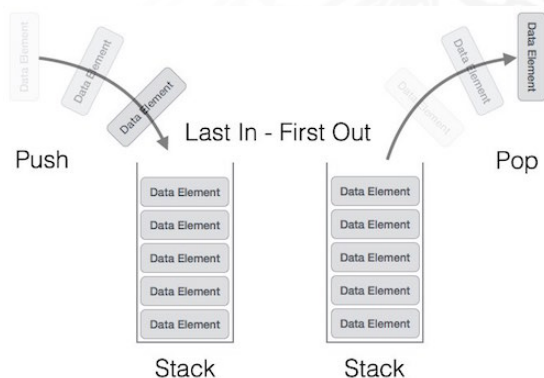
- **Stack**: è organizzata in segmenti che chiameremo *record di attivazione* di un metodo che vengono allocati in corrispondenza di ogni chiamata di metodo ed eliminati in corrispondenza del ritorno del metodo;
- **Heap**: vengono allocati gli oggetti e gli array.

Per eseguire un metodo, si utilizza lo **stack**

Esecuzione dei metodi: lo stack

- Stack = pila si comporta come una lista in cui l'ultimo elemento ad entrare è il primo ad uscire.

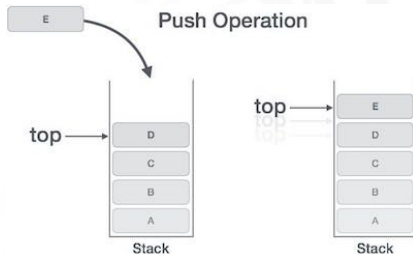
Una pila è una lista LIFO (= last in - first out)



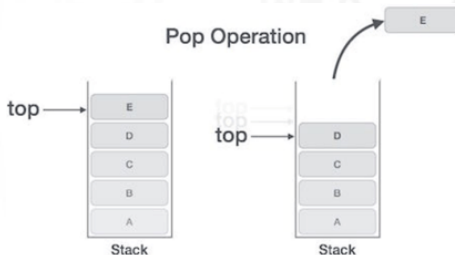
Esecuzione dei metodi: lo stack

Operazioni ammissibili sullo stack:

- *Push*:



- *Pop*:



Record di attivazione

- Lo stack è uno stack di *record di attivazione*;
- ogni record di attivazione contiene i dati necessari per gestire un'invocazione di un metodo:

```
static int raddoppia(int i) {  
    i = i * 2;  
    return i;  
}
```

(da parte del `main`) avrà quindi la forma iniziale

raddoppia	
i	?
risultato	?
ritorno	?

i campi *risultato* e *ritorno* sono usati per la gestione dei metodi da esso invocato

Record di attivazione

```
static int raddoppia(int i) {  
    i = i * 2;  
    return i;  
}
```

(da parte del `main`) avrà quindi la forma iniziale

raddoppia	
i	?
risultato	?
ritorno	?

- *risultato*: se il metodo chiamato restituisce un valore, tale valore viene copiato nel campo risultato del chiamante
- *ritorno*: specifica quale istruzione del metodo deve essere eseguita nel momento in cui il metodo invocato termina la sua esecuzione;

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	?
z	?
risultato	?
ritorno	?

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	3
k	?
risultato	?
ritorno	?

main	
x	3
y	?
z	?
risultato	?
ritorno	❶

Alla invocazione di un metodo:

- si memorizza, nel record di attivazione del chiamante, il punto di rientro del metodo invocato (campo *ritorno*)

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	3
k	?
risultato	?
ritorno	?

main	
x	3
y	?
z	?
risultato	?
ritorno	❶

Alla invocazione di un metodo:

- si crea il record di attivazione del metodo invocato (in questo caso *raddoppia*) con i campi opportuni (in questo caso, i, *k*, *risultato* e *ritorno*)

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	3
k	?
risultato	?
ritorno	?

main	
x	3
y	?
z	?
risultato	?
ritorno	❶

Alla invocazione di un metodo:

- si effettuano le operazioni richieste per la gestione del passaggio dei parametri: valutazione dei parametri attuali e assegnamento dei valori ai parametri formali.

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	3
k	6
risultato	?
ritorno	?

main	
x	3
y	?
z	?
risultato	?
ritorno	❶

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	?
z	?
risultato	6
ritorno	❶

All'esecuzione dell'istruzione return k:

- memorizzazione del risultato nel record di attivazione del chiamante;

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	?
z	?
risultato	6
ritorno	❶

All'esecuzione dell'istruzione return k:

- pop del record di attivazione dell'invocazione di raddoppia;

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	?
z	?
risultato	6
ritorno	❶

All'esecuzione dell'istruzione return k:

- proseguimento dell'esecuzione dal punto indicato nel campo ritorno del record di attivazione del chiamante

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	6
z	?
risultato	6
ritorno	❶

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	6
k	?
risultato	?
ritorno	?

main	
x	3
y	6
z	?
risultato	6
ritorno	❷

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

raddoppia	
i	6
k	12
risultato	?
ritorno	?

main	
x	3
y	6
z	?
risultato	6
ritorno	❷

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	6
z	?
risultato	12
ritorno	❷

Un esempio di esecuzione

```
class Doppio {  
  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }  
  
    public static void main (String[] args) {  
        int x = 3;  
        int y = raddoppia(x); ❶  
        int z = raddoppia(y); ❷  
        System.out.println (y);  
        System.out.println (z);  
    }  
}
```

main	
x	3
y	6
z	12
risultato	12
ritorno	❷

- Al termine dell'esecuzione dell'istruzione di assegnamento, la JVM scrive su stdout i valori delle variabili y e z, e al termine anche il record di attivazione del main viene disallocato dallo stack.