Metodi

Metodi

metodo = un blocco di dichiarazioni e istruzioni con un **nome** ed eventuali **parametri**, che può essere invocato da un altro metodo per eseguire tali istruzioni

```
class Esempio {

    static void saluto () {
        System.out.println("Ciao! ");
    }

    public static void main (String[] args) {
    saluto ();
    } //fine main
} //fine classe
```

Esecuzione: Ciao!

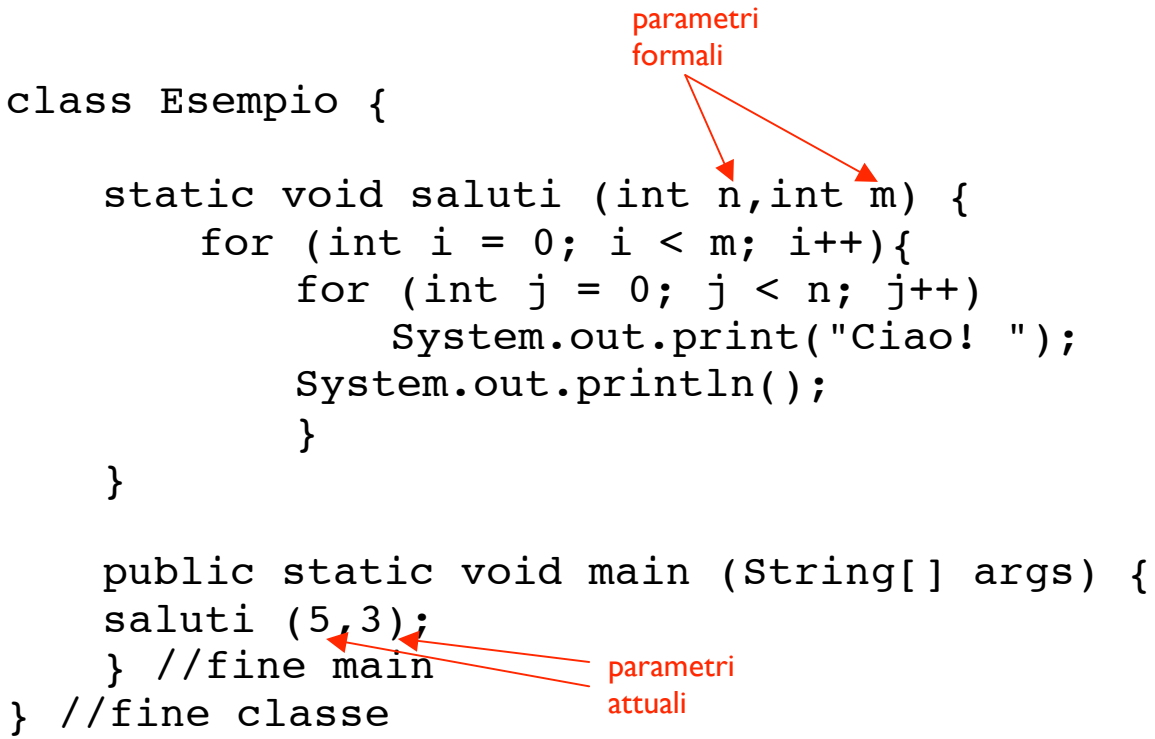
Parametri

Le azioni possono essere parametrizzate

```
class Esempio {

    static void saluti (int m) {
        for (int i = 0; i < m; i++)
        System.out.print("Ciao! ");
    }

    public static void main (String[] args) {
    saluti (5);
    } //fine main
} //fine classe
```

Esecuzione:

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



Esecuzione:

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

Metodi con tipo

Ci sono due tipi di metodi: i **metodi con tipo** e i **metodi void**

Un metodo con tipo è **dichiarato** nel modo seguente:

```
<modificatori> <tipo> <nome> (<parametri>) {
        <dichiarazioni e istruzioni>
        <istruzione return>
}
```

- **modificatori**: public, static, …
- **tipo**: int, char, boolean, double, int[ ], double[ ][ ], …
- **parametri**: come le dichiarazioni di variabili
- **istruzione return**: un'istruzione della forma

  return <espressione>

  dove <espressione> ha lo stesso tipo dichiarato per il metodo

Metodi void

Un metodo void è dichiarato nel modo seguente:

```
<modificatori>  void  <nome> (<parametri>) {
        <dichiarazioni e istruzioni>
}
```

- **modificatori**: public, static, …
- **parametri**: come le dichiarazioni di variabili

Invocazione di un metodo

Restrizione per questa parte del corso: solo metodi **statici**

Due 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 una **espressione** di tipo `int` della forma `m()`
  - in un'altra classe `D`: è una espressione di tipo `int` della forma `C.m()`

  (esempio: `int n = SavitchIn.readLineInt()`)

- **Per i metodi void**

  se un metodo `static void m()` è dichiarato in una classe `C`, allora la sua invocazione

  - in `C`: è una **istruzione** della forma `m();`
  - in un'altra classe `D`: è una istruzione della forma `C.m();`

  (esempio: `System.out.println();`)

Istruzione return

```
class Parametri {

    static int a;
    static int f(int i) {
    if (i == 0)
        return i;
    else
        System.out.println(i);
    }

    public static void main (String[] args) {
        int a = 5;
        a = f(a);
        System.out.println(a);
    } //fine main
} //fine classe
```

```
Errore: Return required at end of int f(int)
```

```
class Parametri {

    static int a;
    static int f(int i) {
    if (i == 0)
        return i;
    else
        return 8;
    }

    public static void main (String[] args) {
        int a = 5;
        a = f(a);
        System.out.println(a);
    } //fine main
} //fine classe
```

Esecuzione:

8

```
class Parametri {

    static int a;
    static void f(int i) {
        i = i + 1;
        return;
        System.out.println(i);
    }

    public static void main (String[] args) {
        int a = 5;
        f(a);
        System.out.println(a);
    } //fine main
} //fine classe
```

```
Errore in compilazione: statement not reached
        System.out.println(i)
```

Un metodo `void`

```
class Parametri {
    public static void raddoppia(int i) {
    i = i * 2;
    System.out.println(i);
    }
    public static void main (String[] args) {
        int numero;
        System.out.println("Scrivere un numero seguito da Invio");
        numero = SavitchIn.readLineInt();
        System.out.println ("Numero prima del raddoppio: " + numero);
        raddoppia(numero);
        System.out.println ("Numero dopo il raddoppio: " + numero);
    } //fine main
} //fine classe
```

Esecuzione:

```
Scrivere un numero seguito da Invio
10
Numero prima del raddoppio: 10
20
Numero dopo il raddoppio: 10
```

Un metodo di tipo `int`

```
class Parametri {

    public static int raddoppia(int i) {
    i = i * 2;
    return i;
    }
    public static void main (String[] args) {
        int numero;
        System.out.println("Scrivere un numero seguito da Invio");
        numero = SavitchIn.readLineInt();
        System.out.println ("Numero prima del raddoppio: " + numero);
        System.out.println ("Numero dopo il raddoppio: " + raddoppia(numero));
    } //fine main
} //fine classe
```

Esecuzione:

```
Scrivere un numero seguito da Invio
10
Numero prima del raddoppio: 10
Numero dopo il raddoppio: 20
```

```
class Parametri {

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

    public static void main (String[] args) {
        System.out.println("Scrivere un numero seguito da Invio");
        numero = SavitchIn.readLineInt();
        System.out.println ("Numero prima del raddoppio: " + numero);
        System.out.println ("Numero dopo il raddoppio: " + raddoppia(numero));
    } //fine main
} //fine classe
```

Esecuzione:

```
Can't make static reference to method int raddoppia(int) in class
Parametri.
        System.out.println ("Numero dopo il raddoppio: " +
raddoppia(numero));
```

```
class Parametri {

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

    public static void main (String[] args) {

        System.out.println("Scrivere un numero seguito da Invio");
        numero = SavitchIn.readLineInt();
        System.out.println ("Numero prima del raddoppio: " + numero);
        System.out.println ("Numero dopo il
                    raddoppio: " + raddoppia(numero));
    } //fine main
} //fine classe
```

Esecuzione:

```
Can't make a static reference to nonstatic variable numero in
class Parametri.
       numero = SavitchIn.readLineInt();
```

Un metodo che usa un parametro con lo stesso nome di una variabile locale ad un altro blocco

```
class ProvaParametri {
    static int i;
    static int raddoppia(int i) {
        i = i * 2;
        return i;
        }
    public static void main (String[] args) {
        System.out.println("Scrivere un numero seguito da Invio");
        i = SavitchIn.readLineInt();
        System.out.println ("Numero prima del raddoppio: " + i);
        System.out.println ("Numero dopo il raddoppio: " + raddoppia(i));
    } //fine main
} //fine classe
```

Esecuzione:

```
Scrivere un numero seguito da Invio
10
Numero prima del raddoppio: 10
Numero dopo il raddoppio: 20
```

Un esempio simile

```
class ProvaParametri {
    static int i;

    static void f(int i) {
        i = i+1;
        System.out.println(i);
    }

    public static void main (String[] args) {
    i = 0;
    System.out.println(i);
    f(i);
    System.out.println(i);
    } //fine main
} //fine classe
```

Esecuzione:

0

1

0

Parametri come variabili locali

```
class ProvaParametri {
    static int i;
    static int raddoppia () {
        int j;
        j = i;
        j = j * 2;
        return j;
        }
    public static void main (String[] args) {
    System.out.println("Scrivere un numero seguito da Invio");
    i = SavitchIn.readLineInt();
    System.out.println ("Numero prima del raddoppio: " + i);
        System.out.println ("Numero dopo il raddoppio: " + raddoppia());
    } //fine main
} //fine classe
```

Esecuzione:

```
Scrivere un numero seguito da Invio
10
Numero prima del raddoppio: 10
Numero dopo il raddoppio: 20
```

Duplicazione di variabili locali

```
class ProvaParametri {

    static int i;
    static void f(int x) {
    int x;
    x = 1;
    System.out.println(x);
    }
    public static void main (String[] args) {

        System.out.println("Scrivere un numero
                                seguito da Invio");
        i = SavitchIn.readLineInt();
        f(i);
        System.out.println ("Numero dopo il
                                raddoppio: " + i);
    } //fine main
} //fine classe
```

Errore in compilazione

Variable 'x' is already defined in this method.

Blocchi e visibilità delle variabili

In questo blocco si cerca di accedere ad una variabile non più visibile perché definita in un blocco esterno:

```
{
int x = 2;
}
System.out.println(x);
```

Errore in compilazione:

```
Undefined variable: x
```

In quest'altro esempio invece si ridichiara una variabile già visibile:

```
{
int x = 2;
{int x = 3;
System.out.println(x);
}
System.out.println(x);
}
```

Errore in compilazione:

```
Variable 'x' is already defined in this method.
```

Il codice

```
{
    int x = 1;
    System.out.println(x);
}
{
    int x = 2;
    System.out.println(x);
}
```

è legittimo. La sua esecuzione dà, come atteso:

```
1
2
```

Quindi, è **possibile** ridefinire un identificatore dichiarato in un blocco esterno

Blocchi e cicli for

```
int i = 10;
System.out.println(i);

for (int i = 0; i < 5; i++) {
System.out.print(i);
}
```

dà un errore in compilazione, mentre

```
for (int i = 0; i < 5; i++) {
System.out.print(i);
}
System.out.println();
int i = 10;
System.out.println(i);
```

esegue correttamente:

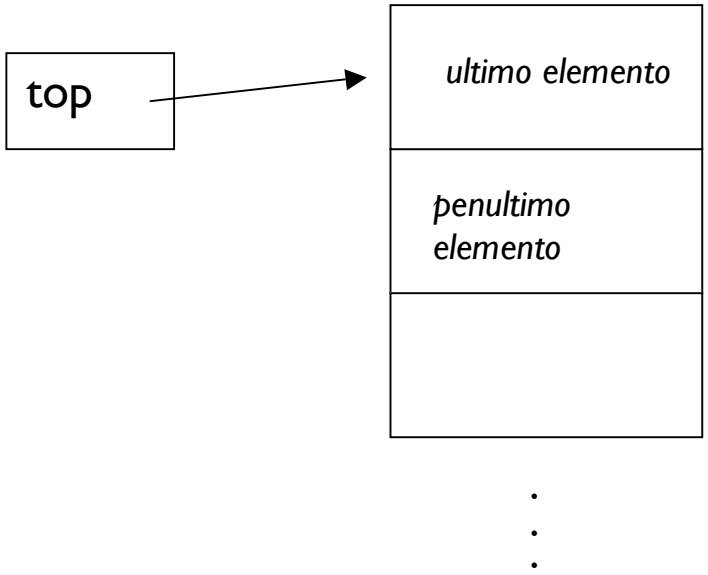
```
01234
10
```

Esecuzione dei metodi — lo stack

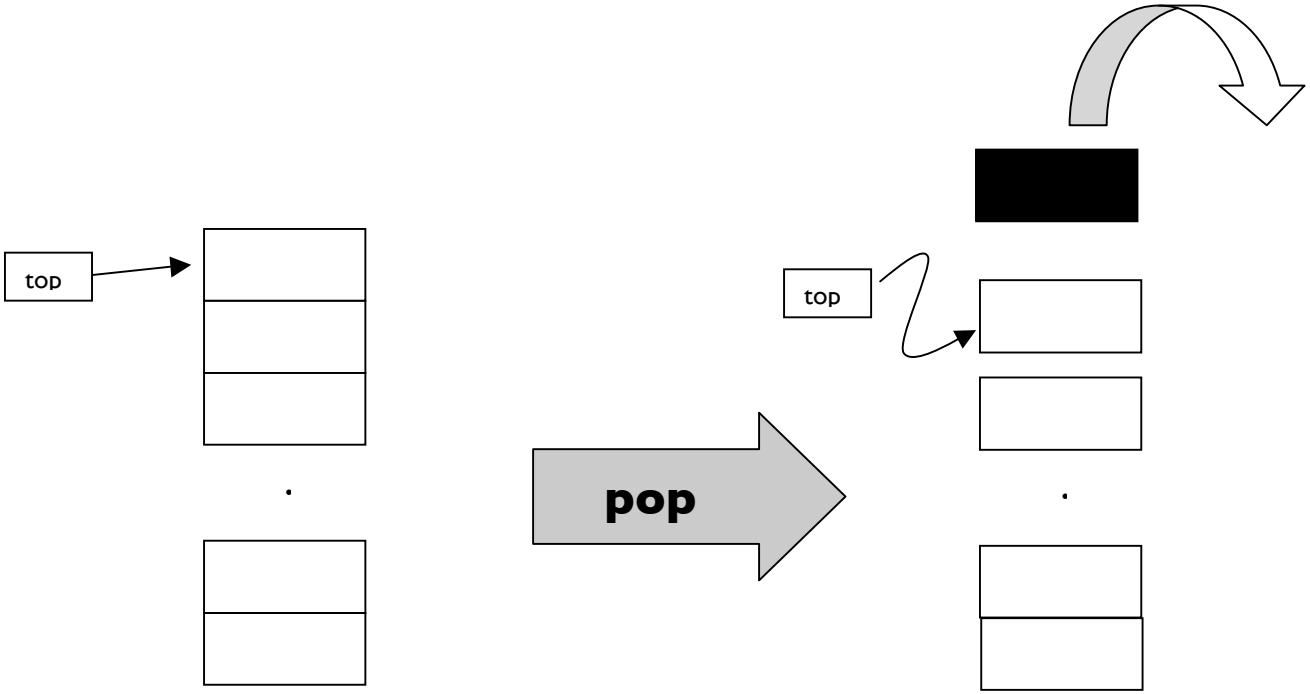
Per eseguire un metodo, si utilizza quella parte della memoria statica detta **stack**.

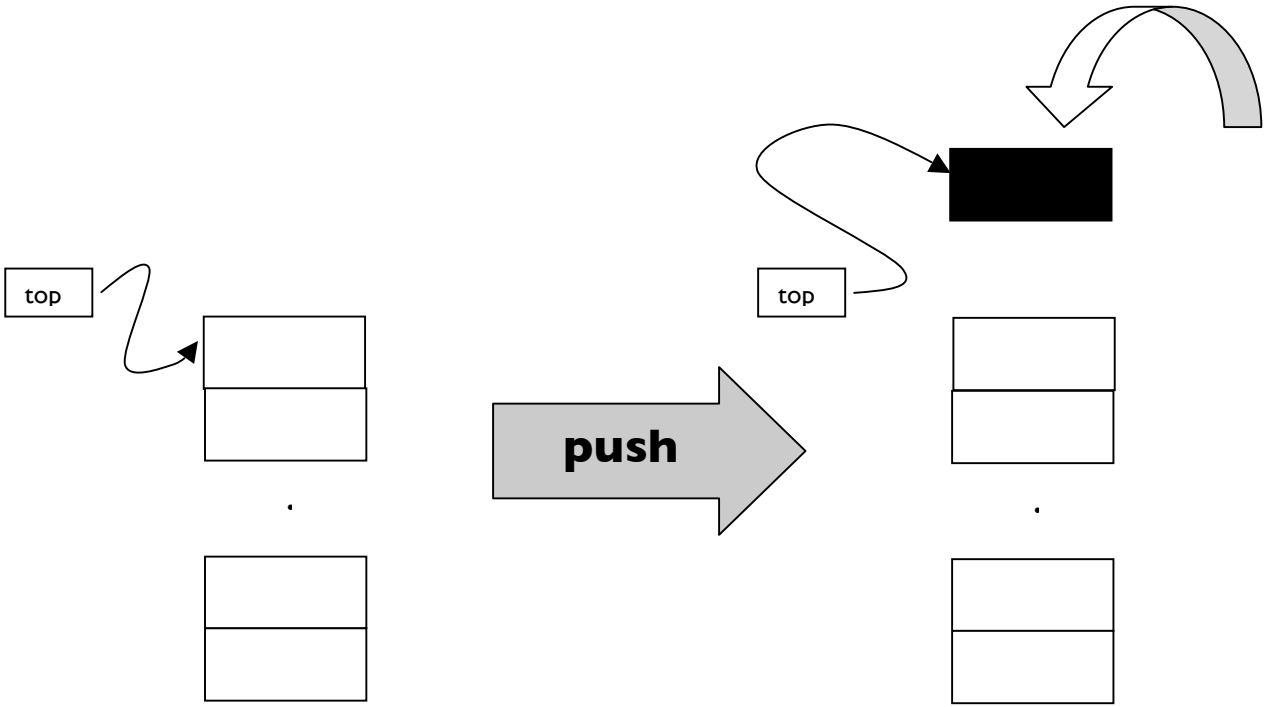
Lo stack (= **pila**) si comporta come una lista in cui l'ultimo elemento ad entrare è il primo ad uscire. Quindi

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



Operazioni ammissibili su stack




top
push
top

Record di attivazione

Lo stack è uno stack di **record di attivazione**, dove ciascun record di attivazione contiene i dati necessari per gestire un'invocazione di un metodo:

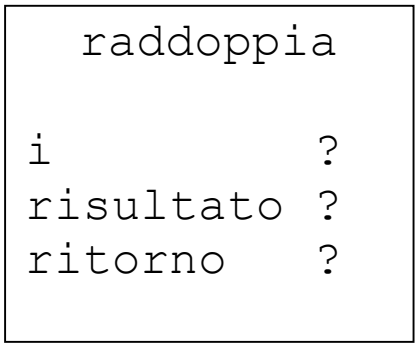
- Informazioni relative all'invocazione del metodo:
  - Per ricevere i dati dei metodi eventualmente invocati dal metodo
  - Per effetture correttamente il rientro dai metodi richiamati dal metodo
- Dati del metodo:

Le variabili locali
I parametri formali, inizializzati con i valori dei parametri attuali

Un record di attivazione per un'invocazione del metodo

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

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

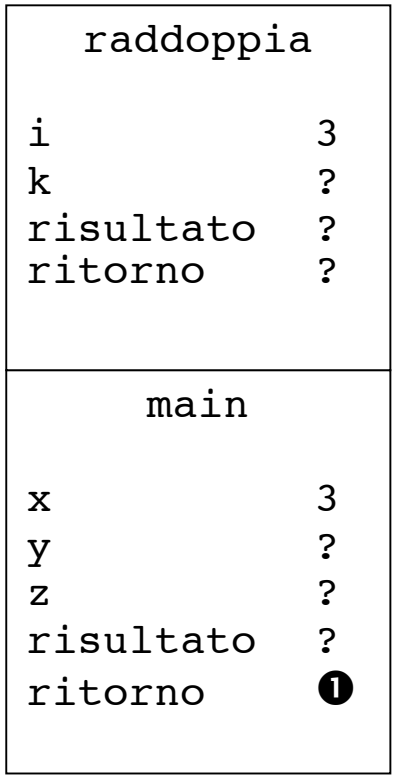


Un esempio di esecuzione

```
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	?

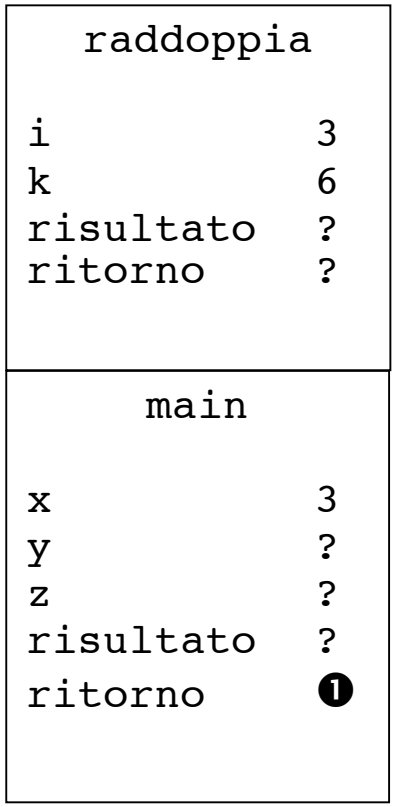
```
class Doppio {

    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);
    }
}
```



Alla invocazione di un metodo:

- si memorizza, **nel record di attivazione del chiamante**, il punto di rientro del metodo invocato (campo `ritorno`)
- 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`)
- si effettuano le operazioni richieste per la gestione del passaggio dei parametri: valutazione dei parametri attuali e assegnamento dei valori ai parametri formali

```
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);
    }
}
```



```
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

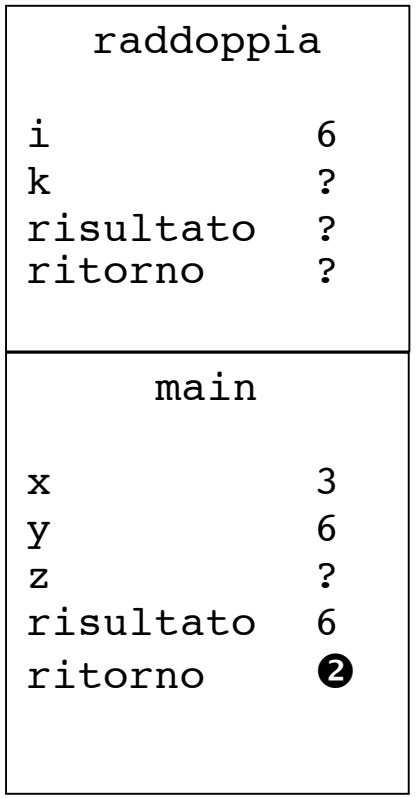
All'esecuzione dell'istruzione `return k`:

- memorizzazione del risultato **nel record di attivazione del chiamante**
- pop del record di attivazione dell'invocazione di `raddoppia`
- proseguimento dell'esecuzione dal punto indicato nel campo `ritorno` del record di attivazione del chiamante

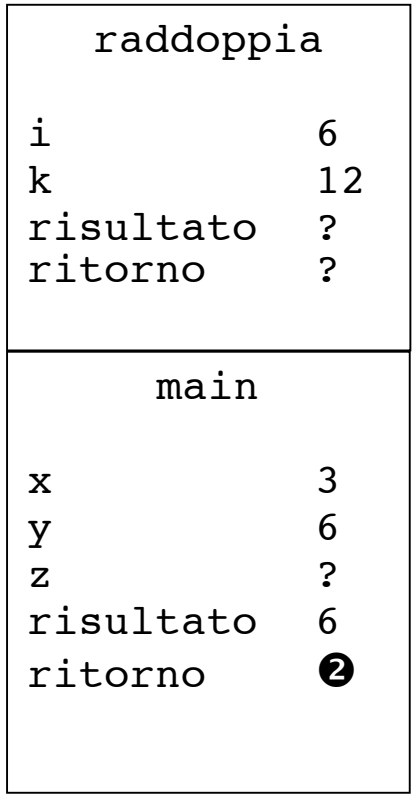
```
class Doppio {

    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

```
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);
    }
}
```



```
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);
    }
}
```



```
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

```
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 Java Virtual Machine scrive su stdout i valori delle variabili `y` e `z`, e al termine anche il record di attivazione del main viene disallocato dallo stack.