

Università di Torino – Scuola di Scienze della Natura
Corso di Studi in Informatica

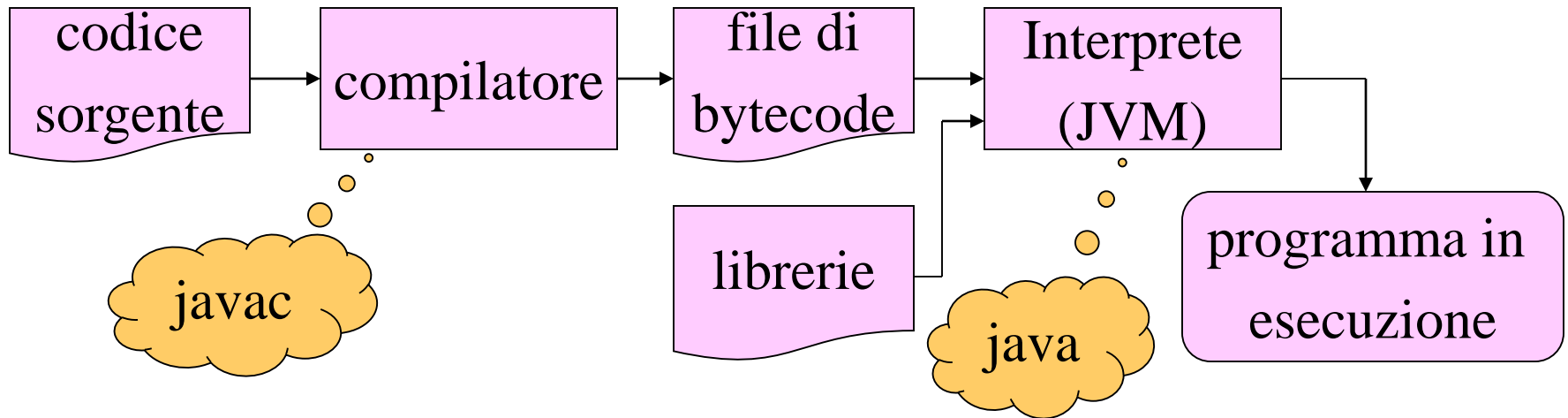
da **Programmazione II - corso B**

prof. Liliana Ardissono

Metodi e gestione memoria della macchina virtuale

Richiami a Java

- Compilazione programmi scritti in Java
 - Dato codice sorgente: MiaClasse.java
 - Fornisce bytecode (eseguibile su macchine fisiche diverse, es: windows e unix): MiaClasse.class



Memoria di JVM - I

- Organizzata in 3 parti principali
 - **Memoria statica**
 - mantiene costanti e variabili statiche (variabili di tipo semplice e riferimenti a oggetti)
 - area stabile, non varia durante esecuzione di programma
 - **Stack**
 - mantiene record di attivazione di metodi; per ciascun record di attivazione, mantiene variabili dei metodi (variabili di tipo semplice e riferimenti a oggetti)
 - gestito come una **pila** (LIFO)
 - **Heap**
 - mantiene dati creati dinamicamente (oggetti)
 - quando dati non sono più indirizzati dal programma, **Garbage Collector** libera la memoria da essi occupata per il riciclo. Il Garbage Collector è un programma SW che agisce in background durante l'esecuzione dei programmi affinché il programmatore non debba preoccuparsi di rilasciare le aree di memoria dismesse in modo esplicito.

Classi e Metodi (statici)

```
public class Esempio {
```

```
    public 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);
```

```
        // Esempio.saluti(5, 3);
```

```
    } //fine main
```

```
} //fine classe
```

Parametri formali

Tipo di output (void)

Parametri attuali

Esecuzione:

Ciao! Ciao! Ciao! Ciao! Ciao!

Ciao! Ciao! Ciao! Ciao! Ciao!

Ciao! Ciao! Ciao! Ciao! Ciao!

Esecuzione di metodi – record di attivazione - I

Record di attivazione (o frame) di un metodo: contiene i dati necessari per gestire l'esecuzione di un metodo:
l'allocazione di un record di attivazione nello stack avviene al momento in cui il metodo viene invocato.

Il record di attivazione di un metodo contiene:

- I parametri formali, inizializzati con i valori dei parametri attuali
- Le variabili locali del metodo
- Il risultato di ritorno per raccogliere il risultato dei metodi invocati dal metodo
- L'indirizzo di ritorno per effettuare correttamente il rientro dall'esecuzione di metodi richiamati dal metodo

Stack – allocazione di frame di un metodo

- Avviene al momento in cui il metodo viene invocato
 - Si alloca spazio al top dello stack
 - NB: variabili locali e parametri di un metodo vivono solo durante l'esecuzione del metodo stesso (mentre il suo record di attivazione sta sullo stack)

Esecuzione di metodi – record di attivazione - II

```
public class Doppio { /* restituisce il doppio del valore in input */  
    public static int raddoppia(int i) {  
        int k = i * 2;  
        return k;  
    }
```

Restituisce risultato
al chiamante

```
public static void main (String[] args) {  
    int x = 3;  
    int y = raddoppia(x);  
    int z = raddoppia(y);  
    System.out.println (y);  
    System.out.println (z);  
} //fine main  
} //fine classe
```

STACK

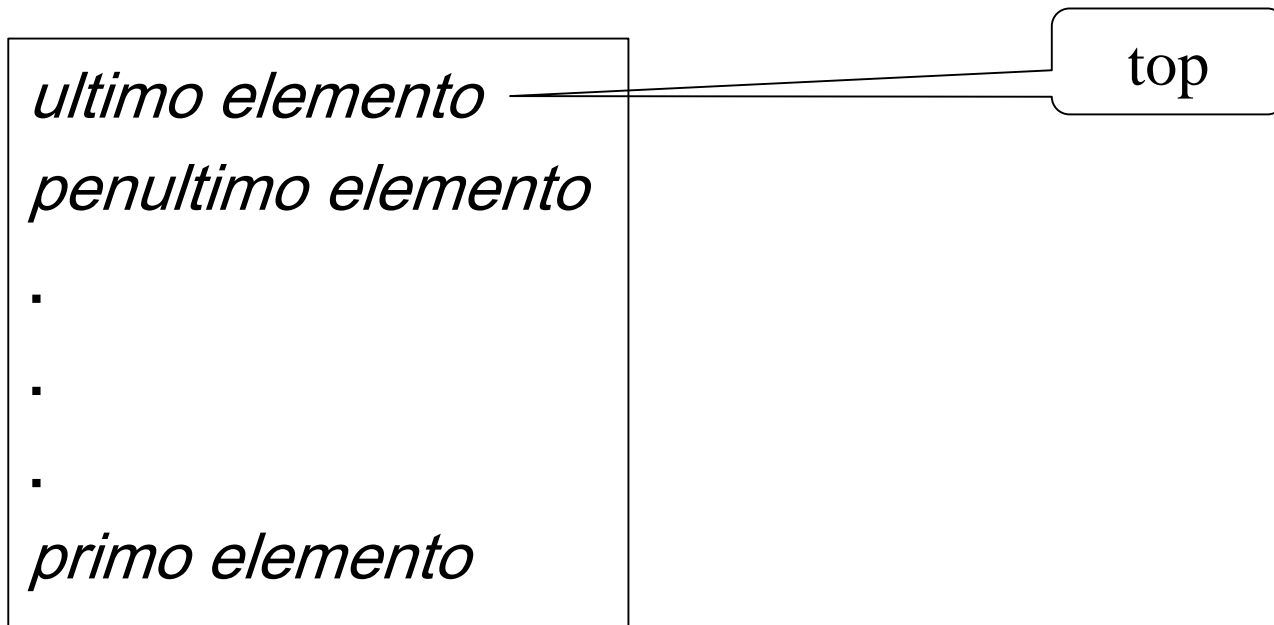
<i>main</i>	
args	null
x	?
y	?
z	?
risultato	?
ritorno	?

Esecuzione di metodi – lo stack

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

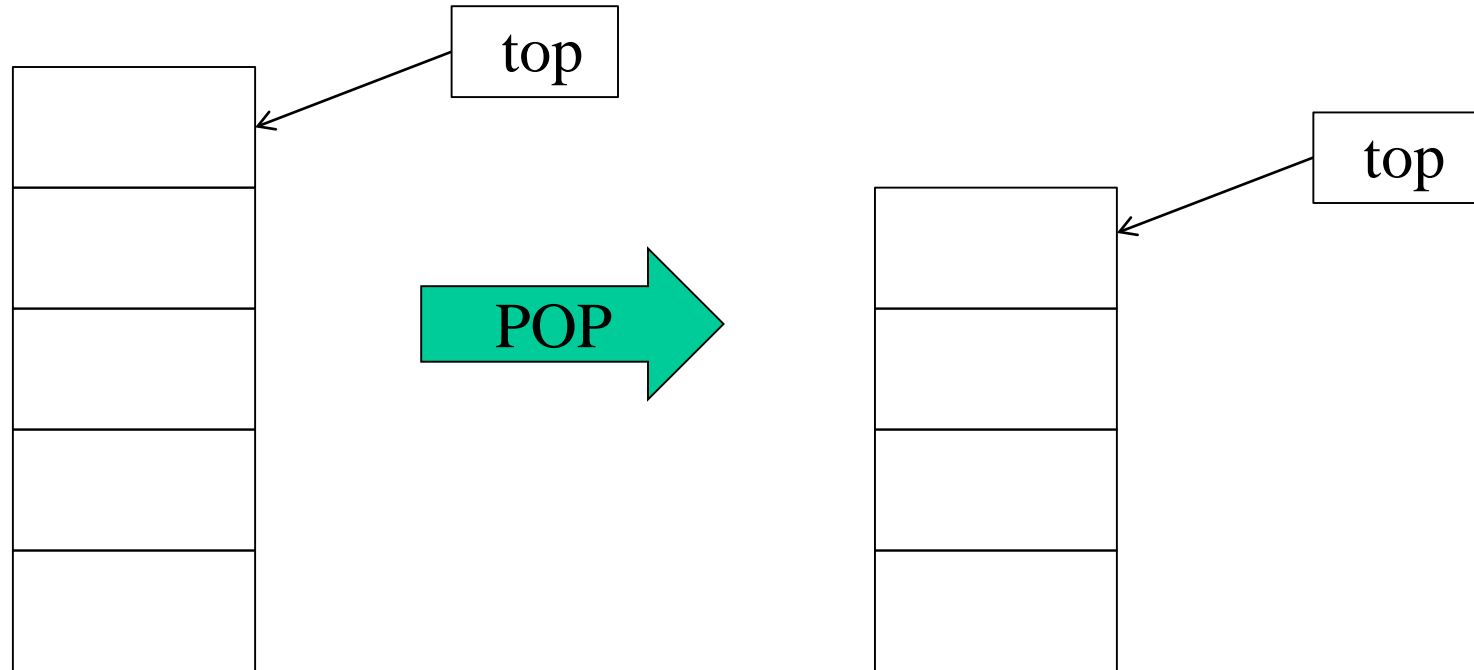
Lo stack (= **pila**) si comporta come una lista in cui l'ultimo elemento ad entrare è il primo ad uscire. Quindi

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



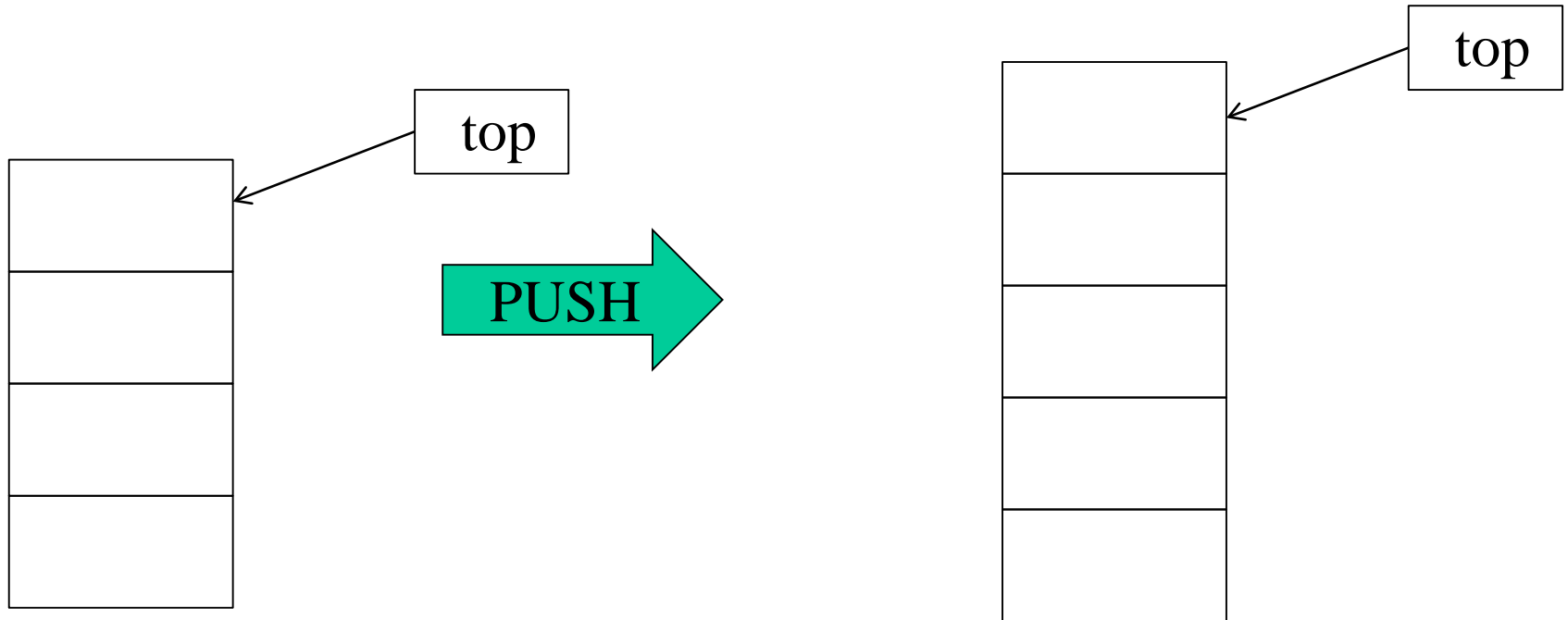
Operazioni ammissibili su uno stack - POP

POP: estrae l'elemento al top della pila



Operazioni ammissibili su uno stack - PUSH

PUSH: inserisce elemento al top della pila



Metodi – esecuzione - I

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

STACK

<i>main</i>	
args	null
x	3
y	?
z	?
risultato	?
ritorno	?

Metodi – esecuzione - II

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



STACK

<i>raddoppia</i>	
i	3
k	?
risultato	?
ritorno	?

<i>main</i>	
args	null
x	3
y	?
z	?
risultato	?
ritorno	❶

Metodi – esecuzione - III

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



STACK

<i>raddoppia</i>	
i	3
k	6
risultato	?
ritorno	?

<i>main</i>	
args	null
x	3
y	?
z	?
risultato	?
ritorno	❶

Metodi – esecuzione - IV

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

STACK

<i>main</i>	
args	null
x	3
y	?
z	?
risultato	6
ritorno	❶


Metodi – esecuzione - V

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

STACK

<i>main</i>	
args	null
x	3
y	6
z	?
risultato	6
ritorno	❶

Metodi – esecuzione - VI



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

STACK

<i>raddoppia</i>	
i	6
k	?
risultato	?
ritorno	?

<i>main</i>	
args	null
x	3
y	6
z	?
risultato	6
ritorno	❷

Metodi – esecuzione - VII

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

STACK

<i>raddoppia</i>	
i	6
k	12
risultato	?
ritorno	?

<i>main</i>	
args	null
x	3
y	6
z	?
risultato	6
ritorno	❷

Metodi – esecuzione - VIII

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

STACK

<i>main</i>	
args	null
x	3
y	6
z	?
risultato	12
ritorno	❷

Metodi – esecuzione - IX

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

STACK

<i>main</i>	
args	null
x	3
y	6
z	12
risultato	12
ritorno	❷

Metodi – passaggio di parametri di tipo oggetto (per ora solo array) - I

Riferimenti come parametri: si possono passare riferimenti ad oggetti, come gli array, come parametro (l'indirizzo viene copiato nel parametro, come valore). In tal caso, il metodo opera direttamente sull'elemento a cui il riferimento punta (l'array) e lo può modificare. Esempio:

```
public class ProvaParametri {  
  
    public static void modifica(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            b[i] = i+3;  } }  
  
    public static visualizza(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            System.out.println(b[i]); } }  
}
```

Metodi – passaggio di parametri oggetto - II

```
public static void main (String[] args) {  
    int[] a = new int [5];  
    for (int i = 0; i<a.length; i++)  
        a[i] = 0;  
    modifica(a); // qui il metodo modifica l'array  
                // a passato come parametro  
    visualizza(a); // qui il metodo accede  
                  // all'array per visualizzare dati  
} //fine main  
} //fine classe
```

esecuzione

3
4
5
6
7

Metodi – passaggio di parametri oggetto - III

```
public class ProvaParametri {  
  
    public static void modifica(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            b[i] = i+3; } }  
  
    public static visualizza(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            System.out.println(b[i]); } }  
  
    public static void main (String[] args) {  
        int[] a = new int [5];  
        for (int i = 0; i<a.length; i++)  
            a[i] = 0;  
        modifica(a);  
        visualizza(a); }  
} //fine classe
```

STACK

main	
args	null
i	5
a	
risultato	?
ritorno	?

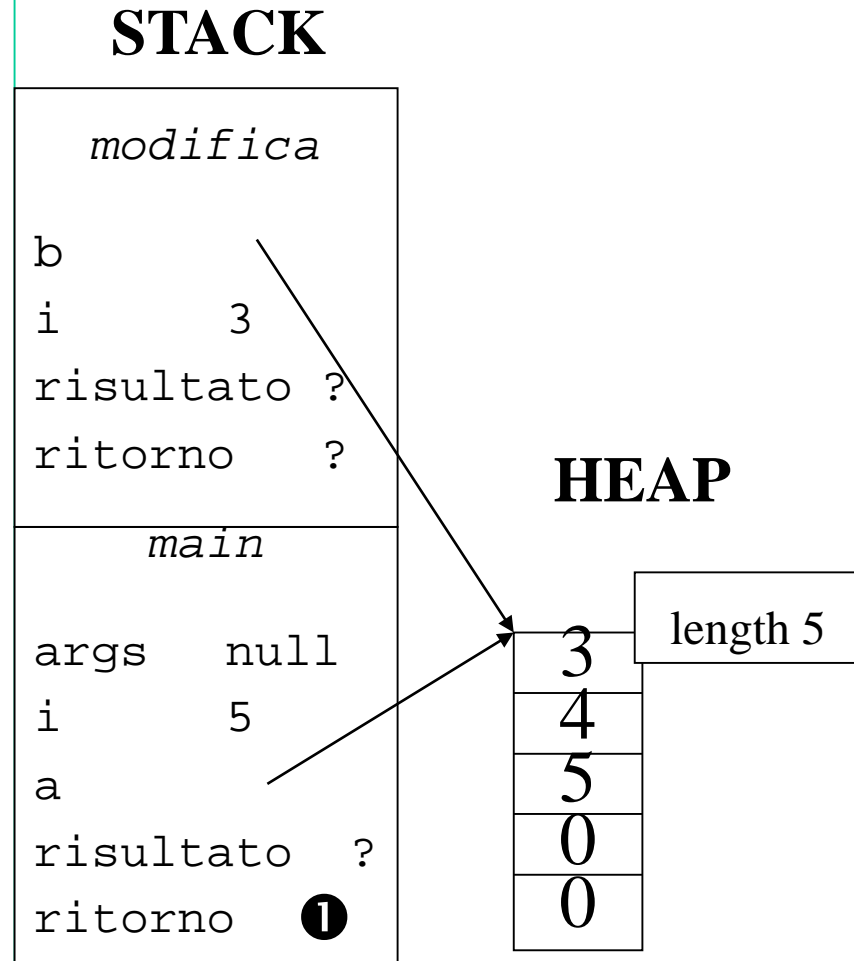
HEAP

0	length 5
0	
0	
0	
0	



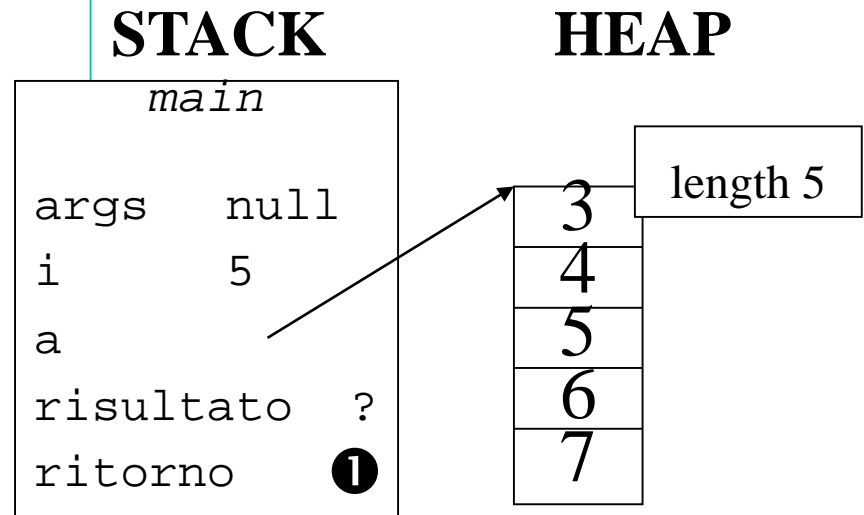
Metodi – passaggio di parametri complessi - III

```
public class ProvaParametri {  
  
    public static void modifica(int[] b) {  
        ➡ for (int i = 0; i < b.length; i++) {  
            b[i] = i+3; } }  
  
    public static visualizza(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            System.out.println(b[i]); } }  
  
    public static void main (String[] args) {  
        int[] a = new int [5];  
        for (int i = 0; i<a.length; i++)  
            a[i] = 0;  
  
        ❶ modifica(a);  
        visualizza(a); }  
} //fine classe
```



Metodi – passaggio di parametri oggetto - IV

```
public class ProvaParametri {  
  
    public static void modifica(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            b[i] = i+3; } }  
  
    public static visualizza(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            System.out.println(b[i]); } }  
  
    public static void main (String[] args) {  
        int[] a = new int [5];  
        for (int i = 0; i<a.length; i++)  
            a[i] = 0;  
  
        ➡ modifica(a);  
        visualizza(a); }  
} //fine classe
```



Gestione della memoria - oggetti

```
public class Point {  
    private static int numPoints = 0; // per memorizzare quanti punti  
                                     // sono stati creati. La dichiaro private  
    private int x;                   // per evitare violazioni del suo valore  
    private int y;                   // (information hiding)  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
        numPoints++; // memorizzo che c'e' una nuova istanza di punto  
    }  
    public static getNumPoints() { return numPoints; }  
    public int getX() { return this.x; }  
    public int getY() { return this.y; }  
    public boolean equals(Point pt) {  
        return this.x==pt.x && this.y==pt.y; }  
} //fine classe
```

Non this.numPoints!!
Metodo statico – usa
var statica

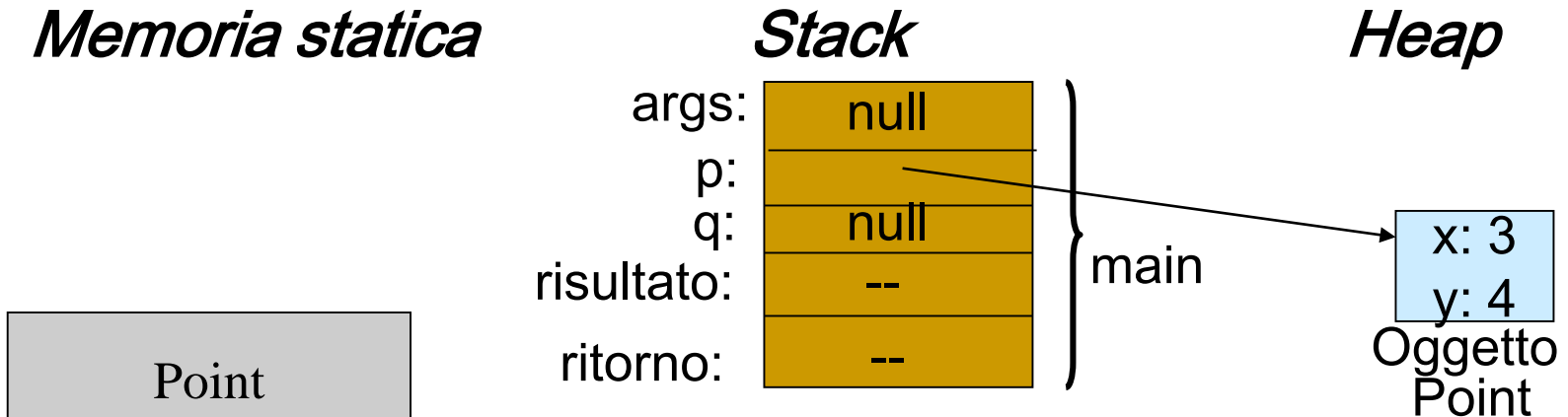
Esempio - II

```
public class PointApp {  
    public static void main(String[] args) {  
        Point p = new Point(3, 4);  
        Point q = new Point(2, 5);  
  
        System.out.println("Ho creato n. " +  
            Point.getNumPoints() + " oggetti Point!");  
  
        System.out.println(p.equals(q));  
    }  
}
```

Invoca il metodo equals() di Point sull'oggetto a cui fa riferimento p. Si passa q come parametro attuale del metodo

Esempio – stack e heap

- Subito dopo la creazione del primo oggetto Point:
Point p = new Point(3,4);

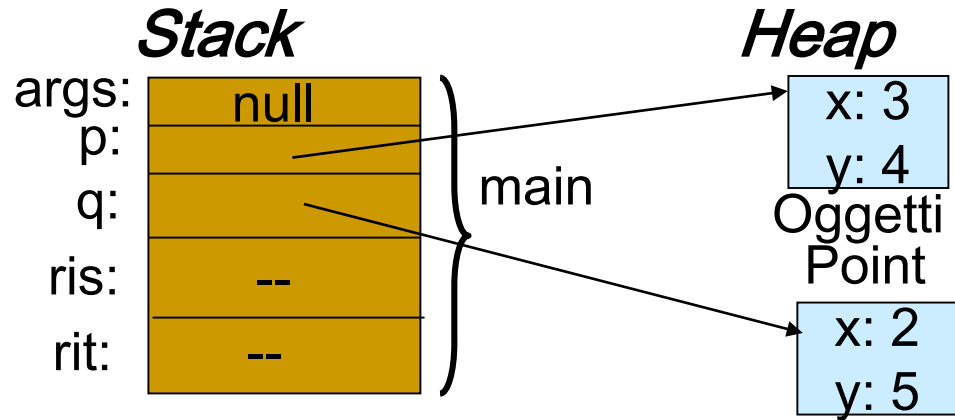
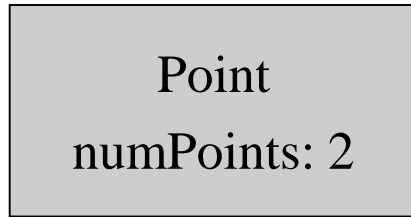


```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ System.out.println(p.equals(q)); }  
$
```

Esempio – stack e heap

- Subito dopo la creazione del secondo oggetto Point:
Point q = new Point(2,5);

Memoria statica



```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ System.out.println(p.equals(q)); }  
$
```

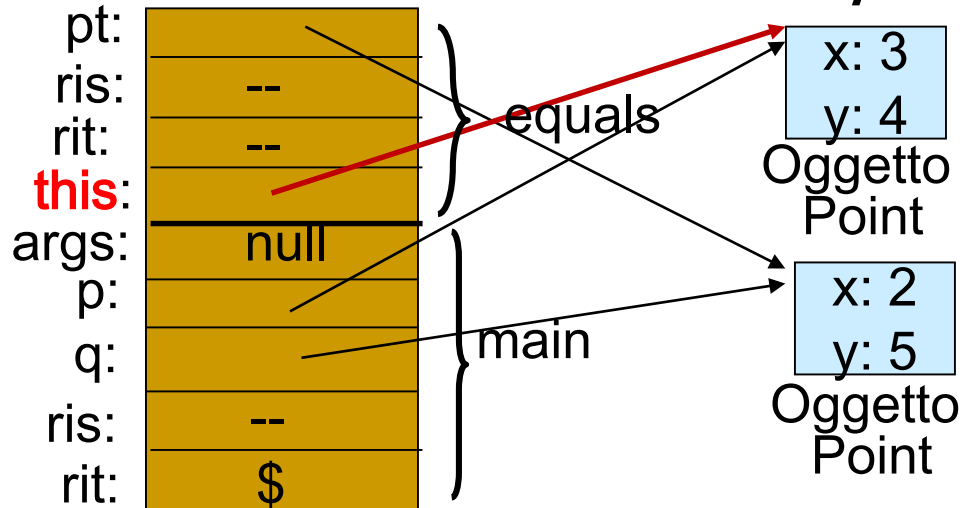
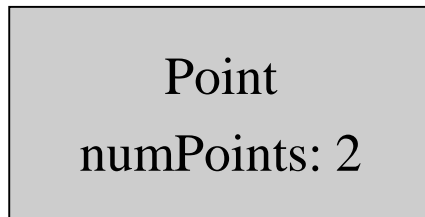
Esempio – stack e heap

- Al momento della valutazione dell'argomento di `System.out.println(p.equals(q))`:

Memoria statica

Stack

Heap



```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ System.out.println(p.equals(q)); }  
$
```

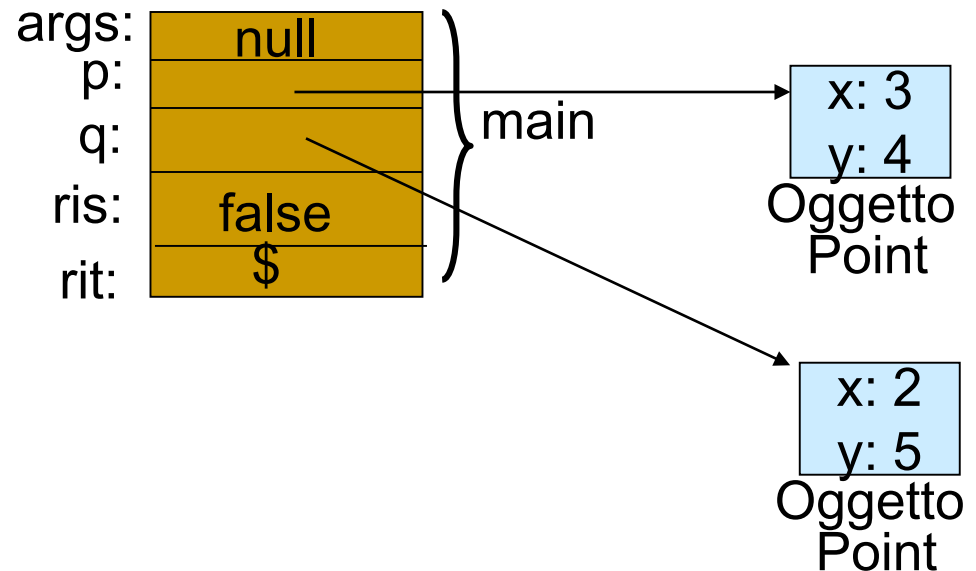
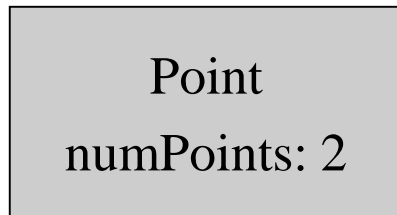
Esempio – stack e heap

- Una volta terminato p.equals(q);

Memoria statica

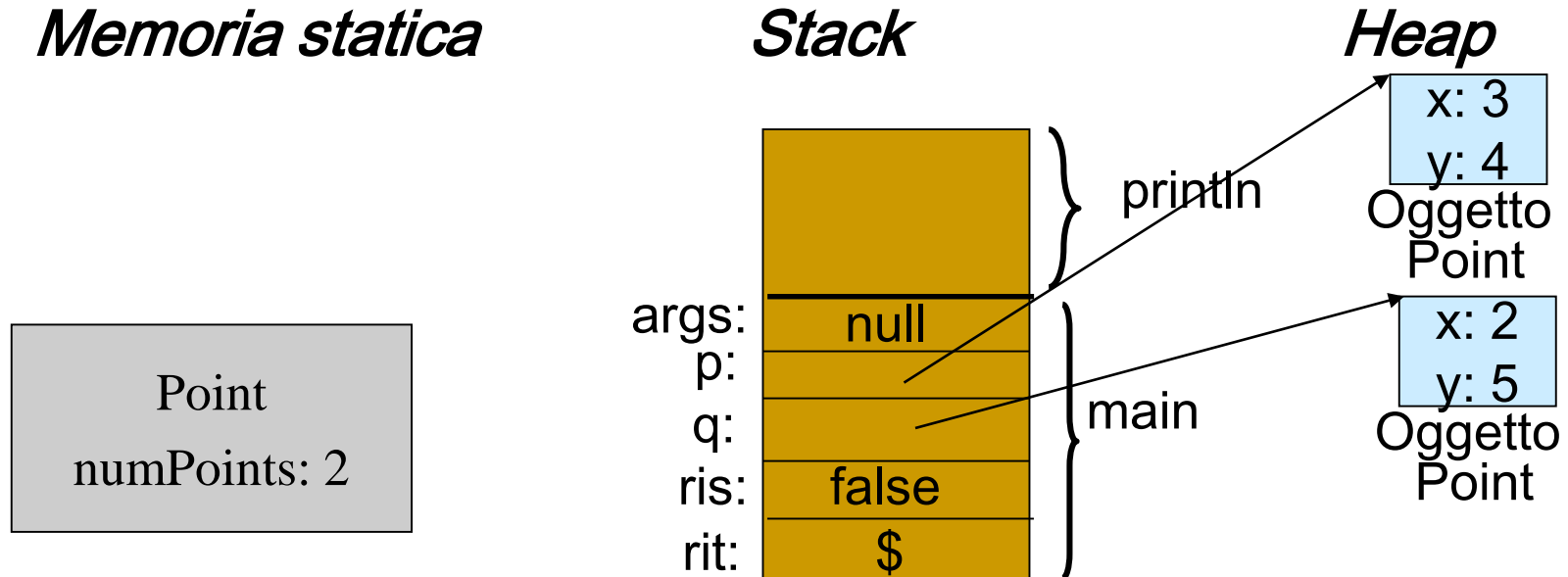
Stack

Heap



Esempio – stack e heap

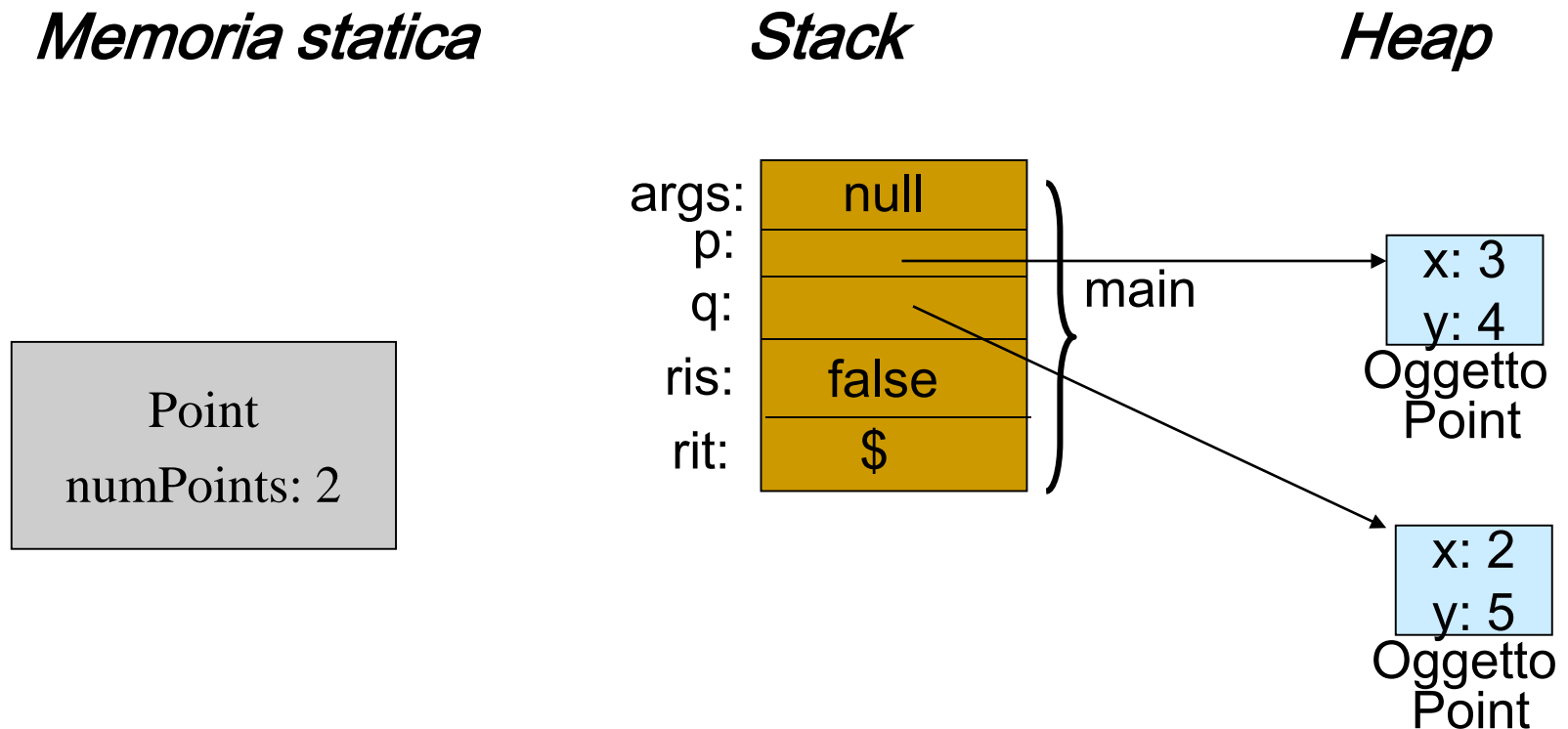
- Durante la `System.out.println()`;



```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ System.out.println(p.equals(q)); }  
$
```

Esempio – stack e heap

- Al termine della `System.out.println();`



Esempio – stack e heap

- Dopo il termine della `System.out.println()` termina anche il main e tutte le aree di memoria relative a PointApp vengono disallocate (garbage collection) per liberare memoria

Memoria statica

Stack

Heap

- Nello stack il campo this di un record di attivazione contiene il riferimento all'oggetto che esegue il metodo a cui è associato il record di attivazione. Serve per poter accedere alle variabili di istanza dell'oggetto. This esiste solo per i metodi di istanza (invocati su oggetti)