



# Programmazione III

Prof.ssa Liliana Ardissono  
Dipartimento di Informatica  
Università di Torino

**Breve ripasso della gestione della memoria nella  
macchina virtuale Java (JVM)**

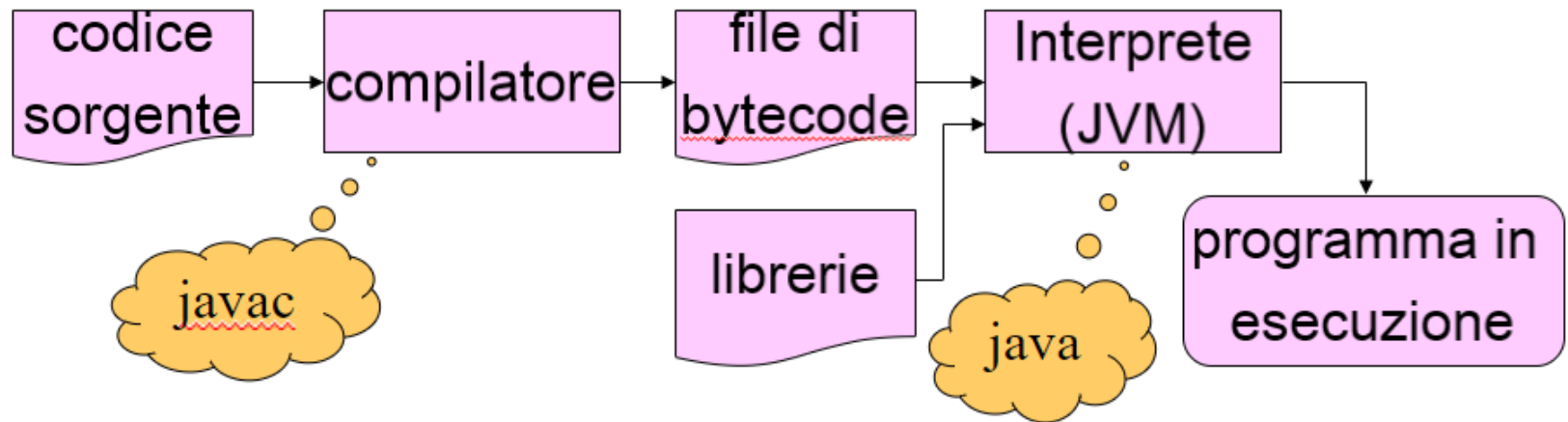


Questa presentazione è distribuita sotto licenza Creative Commons CC BY ND

# Richiami a Java



- Compilazione programmi scritti in Java
  - Dato codice sorgente: MiaClasse.java
  - Fornisce bytecode (eseguibile su macchine fisiche diverse, es: windows e unix) – per esempio, 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)
    - mantiene il codice delle classi
  - **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 i dati creati dinamicamente (oggetti)
    - quando i dati non sono più indirizzati dal programma il **Garbage Collector** libera la memoria da essi occupata per riciclarla. 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) { // n, m: parametri formali  
        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) { // tipo di output: void  
        saluti (5,3); // anche Esempio.saluti(5, 3);  
                      // 5, 3: parametri attuali  
    } //fine main  
} //fine classe
```

**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 del record di attivazione di un metodo



Avviene al momento in cui il metodo viene invocato

- Si alloca spazio al top dello stack
- NB: le variabili locali e i parametri di un metodo esistono solo durante l'esecuzione del metodo stesso (nel periodo temporale in cui il suo record di attivazione sta sullo stack)

# Esecuzione di metodi – record di attivazione II

## situazione dello stack a inizio esecuzione

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



```
public static void main (String[] args) {
```

**STACK**

```
    int x = 3;  
    int y = raddoppia(x);  
    int z = raddoppia(y);  
    System.out.println (z);
```

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

```
    }  
}
```

# 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 (z);
```

```
}
```

```
}
```

**STACK**

*main*

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 (z);  
    }  
}
```

## 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 (z);  
    }  
}
```

## 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 (z);  
}
```

## 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 (z);  
}
```

**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 (z);  
    }  
}
```

## 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 (z);  
    }  
}
```

## 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 (z);
```

```
    }
```

```
}
```

STACK	
main	
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 (z);
```

```
    }
```

```
}
```

STACK	
main	
args	null
x	3
y	6
z	12
risultato	12
ritorno	❷



# Metodi – passaggio di parametri di tipo oggetto - 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 void 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  
}  
}
```

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 void 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); }  
}
```

STACK

main	
args	null
i	5
a	
risultato	?
ritorno	?

HEAP

0	length 5
0	
0	
0	
0	

# Metodi – passaggio di parametri oggetto - IV

```
public class ProvaParametri {
```

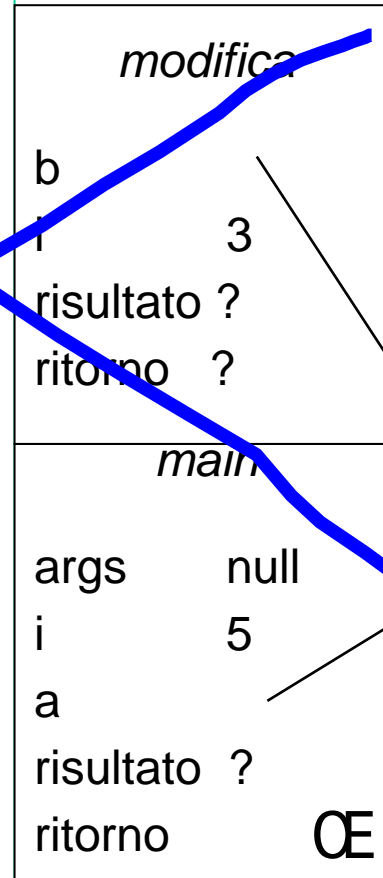
```
    public static void modifica(int[] b) {  
        for (int i = 0; i < b.length; i++) {  
            b[i] = i+3; } }  
    
```

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

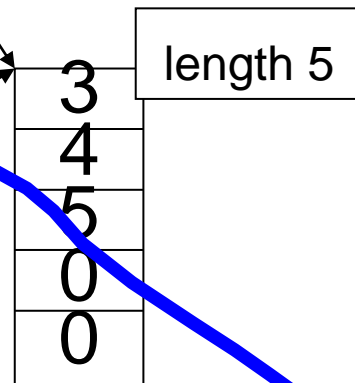
```
    public static void main (String[] args) {  
        int[] a = new int [5];  
        for (int i = 0; i<a.length; i++)  
            a[i] = 0;
```

```
        ① modifica(a);  
        visualizza(a); }  
    }
```

## STACK



## HEAP



# Metodi – passaggio di parametri oggetto - V

```
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); }  
}
```

**STACK**

main	
args	null
i	5
a	
risultato	?
ritorno	

**HEAP**

length 5	
3	
4	
5	
6	
7	

# Variabili statiche e di istanza



- **Variabili di istanza:** servono per memorizzare lo stato degli oggetti – ogni oggetto ha la sua copia della variabile di istanza nella heap (nell'area di memoria dedicata all'oggetto)
- **Variabili di classe, o statiche (static):** c'è una unica copia della variabile per la classe ed è condivisa tra tutti gli oggetti.

Consideriamo i punti nello spazio cartesiano, rappresentati in una applicazione dalla classe Point. Ogni oggetto Point ha le proprie coordinate, che devono essere associate solo a quel punto. Invece, se considero per esempio il numero di punti (istanze) create a partire dalla classe Point, questa deve essere una variabile static per permettere di incrementare il suo valore correttamente ogni volta che l'applicazione crea un nuovo oggetto Point. Vedere prossimo esempio.

numPoints: 3

Point: (125, 34)

Point: (-30, 45)

Point: (20, -10)

# Esempio – stack e heap - I



```
public class Point {
```

```
    private static int numPoints = 0; // numPoints memorizza quanti punti  
    // sono stati creati. La dichiaro private per evitare violazioni del suo
```

```
    private int x; // valore (information hiding)
```

```
    private int y;
```

```
    public Point(int x, int y) {
```

```
        this.x = x; // notare il this per indicare la variabile di istanza
```

```
        this.y = y; // e distinguerla dal parametro (omonimo) del metodo
```

```
        numPoints++; // memorizzo che c'e' una nuova istanza di punto
```

```
        // Non si può scrivere this.numPoints!! E'una variabile statica!!!
```

```
    }
```

```
    public static getNumPoints() { return numPoints; }
```

```
    public int getX() { return x; }
```

```
    public int getY() { return y; }
```

```
    public boolean equals(Point pt) {
```

```
        return x==pt.x && y==pt.y; }
```

```
}
```

# Esempio – stack e heap - 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!");  
  
        p.equals(q);  
    }  
}
```

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

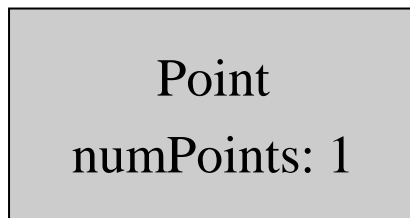


# Esempio – stack e heap - III

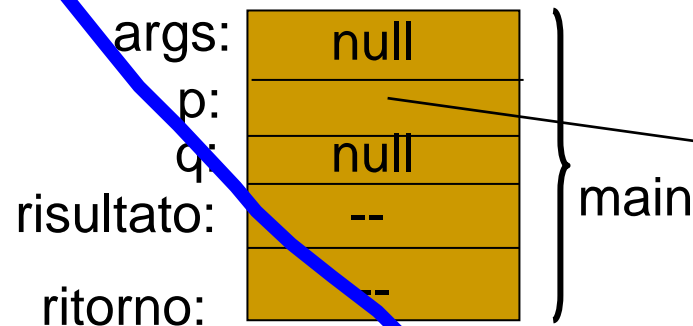


Subito dopo la creazione del primo oggetto Point:  
`Point p = new Point(3,4);`

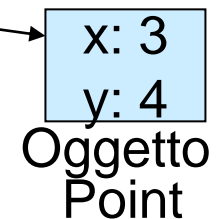
## Memoria statica



## Stack



## Heap



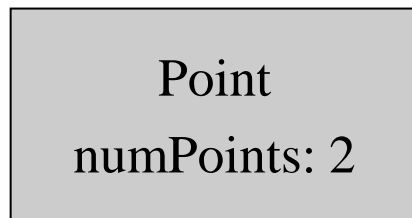
```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ p.equals(q);  
}
```

# Esempio – stack e heap - IV

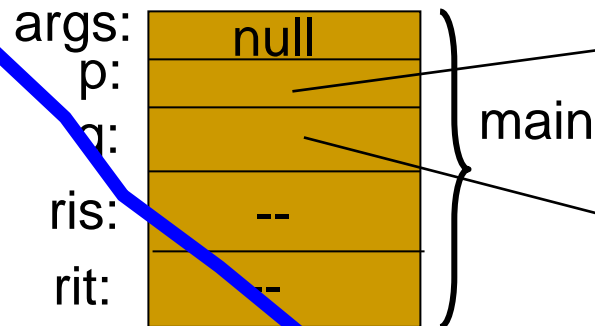


Subito dopo la creazione del secondo oggetto Point:  
Point q = new Point(2,5);

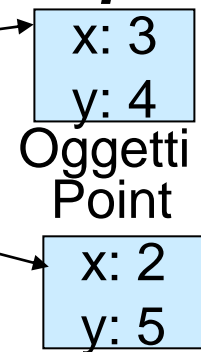
## Memoria statica



## Stack



## Heap



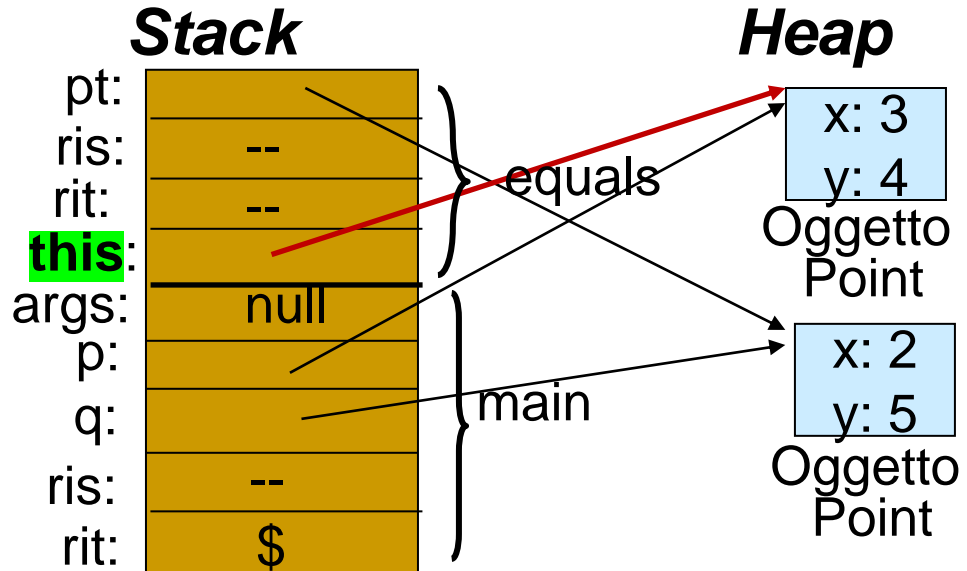
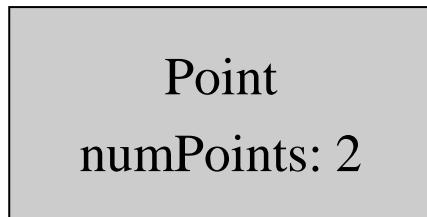
```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    $ p.equals(q);  
}
```

# Esempio – stack e heap - V



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

## Memoria statica



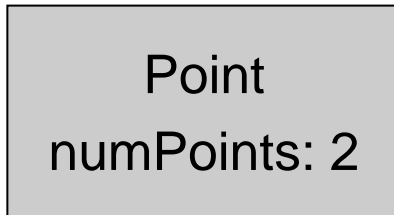
```
public static void main(String[] args) {  
    Point p = new Point(3, 4);  
    Point q = new Point(2, 5);  
    System.out.println("Ho creato n. " +  
        Point.getNumPoints() + " oggetti Point!");  
    → $ p.equals(q);  
}
```

# Esempio – stack e heap - VI

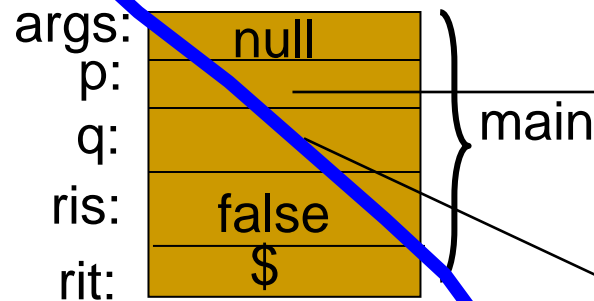


Una volta terminato p.equals(q);

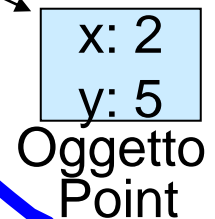
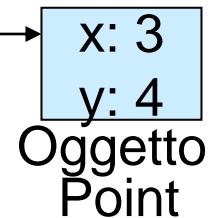
**Memoria statica**



**Stack**



**Heap**

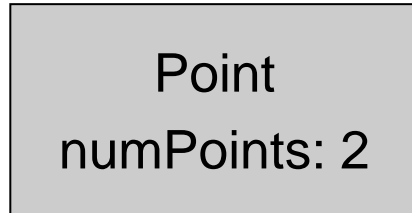


# Esempio – stack e heap - VII

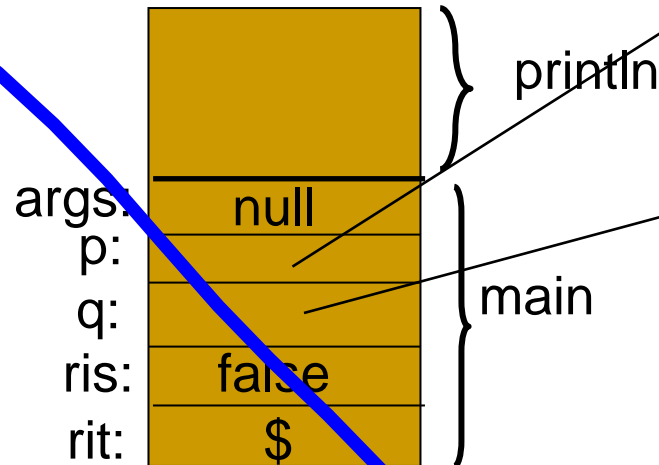


NB: Durante la `System.out.println()`;

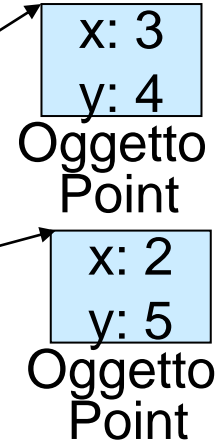
**Memoria statica**



**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!");  
    $ p.equals(q);  
}
```

# Esempio – stack e heap - VIII

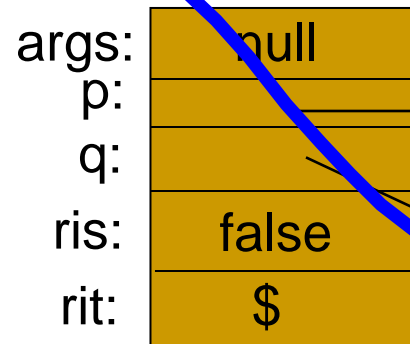


Come sarebbe la memoria se numPoints fosse una variabile di istanza?

## Memoria statica



## Stack



main

## Heap

