Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

# Programmazione I - corso B
# a.a. 2009-10

prof. Viviana Bono

**Blocco 9 – Metodi statici:**
**passaggio parametri, variabili locali, record di attivazione, stack**

# I metodi e lo stack

## Spazi di memoria (o immagazzinamento)

- Registri
  - All'interno del processore => veloci, numero limitato
- Stack
  - Compilatore Java gestisce lo *stack pointer*
  - Compilatore deve conoscere dimensione oggetti su stack
- Heap
  - Tutti gli oggetti Java vi risiedono
  - Compilatore non deve conoscere spazio su heap



## Spazi di memoria (o immagazzinamento, cont.)

- Spazio memoria statico
  - In un "posto fisso"
  - Contiene dati sempre a disposizione durante l'esecuzione del programma (static)
- Spazio memoria costante
  - Valori costanti in una memoria che non viene mai modificata (ROM) o insieme al programma
- Spazio memoria non RAM
  - Per oggetti *persistenti*

## Spazi di memoria (o immagazzinamento, cont.)

Caso speciale: i tipi primitivi

- Non si creano con la *new* (quindi non sono oggetti, che invece stanno nella *heap*)
- Stanno in una variabile "automatica", che non è un *riferimento* ma memorizza direttamente il valore e viene posta sullo *stack*
- short, int, char, byte, boolean, long, float, double



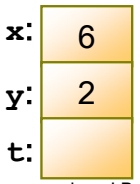
## Un esempio di metodo (funzione)

```
static int myFun(int x, int y) {
  int t = 3*x-4*y;
  return t*t*t;
}
```

```
static int myFun(int x, int y) {
  int t; // anche cosi e` ok
  t = 3*x-4*y;
  return t*t*t;
}
```

## Chiamata di metodo: frame e passaggio parametri

L'"esecuzione" (o meglio, elaborazione) di una *definizione* di metodo-funzione f, ad es. di myFun, consiste semplicemente nella memorizzazione del metodo: non viene eseguita alcuna istruzione. Invece all'istante dell'esecuzione della *chiamata* di f, ad es. myFun(6,2), succede che: in un'area di memoria chiamata *stack* o *pila* viene creato un contenitore chiamato *frame di* f (*o record di attivazione di* f), composto di tre celle, di nomi rispettivamente `x`, `y`, `t`; in `x` e `y` (parametri formali) vengono depositati i valori 6 e 2 (parametri effettivi o argomenti, detti anche *attuali*, trad. scorretta dall'inglese); poi inizia l'esecuzione del corpo della procedura.

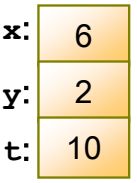



frame di myFun



## Chiamata di metodo: esecuzione del corpo

- viene calcolata l'espressione `3*x - 4*y` il cui significato è:
  3 × contenuto di `x` − 4 × contenuto di `y`
- il suo valore 10 viene depositato nel contenitore `t`:




- viene calcolata l'espressione `t*t*t`, cioè:
  (contenuto di `t`) × (contenuto di `t`) × (contenuto di `t`)
- il suo valore 1000 viene "restituito" al chiamante,
  la procedura termina, il suo frame viene distrutto.

NB In inglese *return* vuol dire *restituisci*.

In altre parole, la definizione di metodo

```
static int myFun(int x, int y) {
  int t = 3*x - 4*y;
  return t*t*t;
}
```

letta in modo antropomorfo, vuol dire:

- il mio nome è myFun;
- posso venire chiamato (o attivato), ma all'atto della chiamata devo ricevere una sequenza di due valori interi, che deposito il primo nel contenitore `x` e il secondo nel contenitore `y`;
- la sequenza di azioni che eseguo quando vengo chiamato è:
  - metto nel contenitore `t` il risultato di

    3 × (contenuto di `x`) − 4 × (contenuto di `y`)
  - restituisco al chiamante il valore che si ottiene facendo

    (contenuto di `t`) × (contenuto di `t`) × (contenuto di `t`)



**Nota**

In realtà il record di attivazione di un metodo contiene anche altre informazioni e altri dati oltre ai parametri e alle variabili locali, ma ciò sarà studiato in corsi successivi.

## Attenzione! non confondere: input e passaggio parametri output e restituzione di un risultato!

Esempio 1 – Il metodo

```
static int myFun(int x, int y) {
  int t = 3*x - 4*y;
  return t*t*t;
}
```

non effettua input-output, cioè non comunica con il mondo esterno, ma solo con la procedura chiamante: da essa riceve gli argomenti (che deposita in `x` e `y`), ad essa restituisce alla fine il risultato.

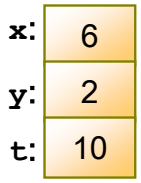
riceve argomenti – restituisce risultato – no input/output



## Attenzione! non confondere: input e passaggio parametri output e restituzione di un risultato!

Esempio 2 – Il metodo:

```
static void myFun() {
  int x = tastiera.nextInt();
  int y = tastiera.nextInt();
  int t = 3*x - 4*y;
  System.out.println(t*t*t);
}
```




non comunica con la procedura chiamante, ma solo col mondo esterno, facendo input (da tastiera) di due valori e alla fine output (sullo schermo) di un risultato.

non prende argomenti – non restituisce nulla – fa input/output

Attenzione! non confondere:
input e passaggio parametri
output e restituzione di un risultato!

Esempio 3 – Il metodo:

```
static int myFun() {
  int x = tastiera.nextInt();
  int y = tastiera.nextInt();
  int t = 3*x - 4*y;
  return(t*t*t);
}
```

legge (cioè fa input da tastiera di) due interi e restituisce al chiamante un risultato.

non prende argomenti – restituisce risultato
fa input – non fa output



Attenzione! non confondere:
input e passaggio parametri
output e restituzione di un risultato!

Esempio 4 – Il metodo:

```
static void myFun(int x, int y) {
  int t = 3*x - 4*y;
  System.out.println(t*t*t);
}
```

prende (dal chiamante) due interi come argomenti e scrive (cioè fa output) sullo schermo un risultato

prende argomenti – non restituisce risultato
non fa input – fa output

## Input-output e passaggio parametri

Attenzione: in generale, l'input-output deve essere tenuto distinto dall'algoritmo vero e proprio. Una procedura che realizza un algoritmo, o che comunque effettua un'elaborazione di dati, di solito è bene che non faccia operazioni di input-output, ma soltanto passaggio parametri e restituzione di risultato.

L'input-output deve essere effettuato, nel caso dei primi semplici esercizi di programmazione, dal main: il metodo myFun deve essere definita come nell'Esempio 1.

Nel caso di strutture-dati complesse l'input-output dovrà essere realizzato da procedure appositamente definite (metodi di input-output). Nelle applicazioni reali tali procedure devono di solito realizzare una vera e propria interfaccia "a finestre" con l'utente.



## L'Esempio 1 completato con il main

```
class Esercizi {

  static int myFun(int x, int y) {
    int t = 3*x - 4*y;
    return t*t*t;
  }

  public static void main(String[] args) {
    int a, b;
    Scanner tastiera = new Scanner(System.in);
    //"tastiera" e` un nome dato dal programmatore
    a = tastiera.nextInt();
    b = tastiera.nextInt();
    System.out.println(myFun(a,b));
  }
}
```

## Dichiarazione e uso di variabili

Attenzione! non confondere:
dichiarazione (o definizione) di variabile
e suo uso successivo!

La scrittura

```
int n, voto;  boolean esameSuperato;
```

stabilisce che nel programma si potranno usare due contenitori `n` e `voto` di tipo `int` ed uno di nome `esameSuperato` di tipo `boolean`. Per usare poi tali contenitori in istruzioni successive, bisogna indicarli SOLO con il loro nome, NON accompagnato dal loro tipo (che è dichiarato una volta per tutte nella dichiarazione)! Esempio:

```
n = 0; esameSuperato = voto > 18;
while(n < 10) {
  n = n + 2;
 ...
```



## Dichiarazione e uso di variabili: inizializzazione

La prima istruzione di assegnazione di una variabile, detta *inizializzazione*, può essere combinata con la sua dichiarazione, in una *dichiarazione con inizializzazione*. Ad esempio, invece di scrivere:

```
int n, voto;
boolean esameSuperato;
n = 0;
voto = 1;
esameSuperato = false;
```

si può, ed anzi è meglio, scrivere:

```
int n = 0;
int voto = 1;
boolean esameSuperato = false;
```

Attenzione: ciò vale solo per l'assegnazione iniziale; nelle assegnazioni successive NON si scrive il tipo! Esempio:

```
voto = 18; n = voto + 2;
```
ecc.

## Variabili locali e variabili globali

- variabili locali: sono le variabili dichiarate all'interno di un metodo (anche del metodo main); esse esistono, al pari dei parametri formali, solo durante l'attivazione del metodo stesso, e sono delle celle nel suo record di attivazione (o frame). Al di fuori del corpo del metodo esse non sono né visibili né definite.

- campi statici (simili alle variabili globali in altri linguaggi): sono le variabili dichiarate fuori dei metodi, ma all'interno di una classe (ricorda: in Java tutto deve essere all'interno di una classe), e qualificate static; esse esistono durante tutta l'esecuzione del programma, e sono visibili dall'interno dei metodi della stessa classe; a certe condizioni esse sono visibili anche dall'interno di metodi di altre classi.



## Esempio 1

```
class Esercizi {
  static int myFun(int x, int y) {
    int t = 3*x - 4*y; //t var.locale di myFun
    return t*t*t;
  }

  public static void main(String[] args) {
    int a, b; // var. locali di main
    Scanner tastiera = new Scanner(System.in);
    // anch'essa var. locale di main
    a = tastiera.nextInt();
    b = tastiera.nextInt();
    System.out.println(myFun(a,b));
  }
}
```

## Esempio 2

```
class Esercizi {
  static int a, b; // var. globali di Esercizi
  static Scanner tastiera = new Scanner(System.in);
  // anch'essa variabile globale

  static int myFun(int x, int y) {
    int t = 3*x - 4*y;
    return t*t*t;
  }

  public static void main(String[] args) {
    a = tastiera.nextInt();
    b = tastiera.nextInt();
    System.out.println(myFun(a,b));
  }
}
```



## Esempio 3

```
class Esercizi {
  static int base = 10; // var. globale
  static int myFun(int x, int y) {
    int t = 3*x - 4*y; // t variabile locale
    return t*t*t;
  }
}

class GestioneCorso {
  static int numPromossi = 0; // var. globale
  static boolean terminato = false;

  static void promuovi(String nome){
    System.out.println(nome + "e' promosso!");
    numPromossi++;
  }
}
```

## Esecuzione di un programma

```
class Esercizi {
  static Scanner tastiera = new
  Scanner(System.in);
  static int a = tastiera.nextInt();

  static int myFun(int x, int y) {
    int t = a*x - 4*y; // t var.locale di myFun
    return t*t;
  }

 public static void main(String[] args) {
    int a, b; // var. locali di main
    a = tastiera.nextInt();
    b = tastiera.nextInt();
    System.out.println(myFun(a,b));
  }} // la seconda } va messa a capo
```



passo 1

```
class Esercizi{
 static Scanner tastiera
```

*Area delle classi* *Stack* *Heap*

classe Esercizi

`tastiera:`

## passo 2

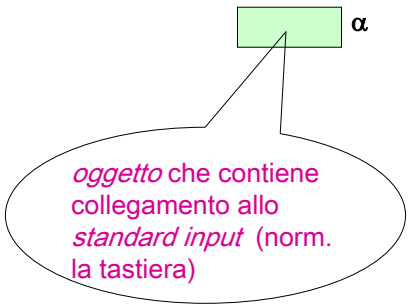
```
class Esercizi{
 static Scanner tastiera=new Scanner(System.in);
```

*Area delle classi* *Stack* *Heap*

classe Esercizi

**tastiera:** α

[α e` un indirizzo della heap]






## passo 3

```
class Esercizi{
 static Scanner tastiera=new Scanner(System.in);
 static int a
```

*Area delle classi* *Stack* *Heap*

classe Esercizi

**tastiera:** α

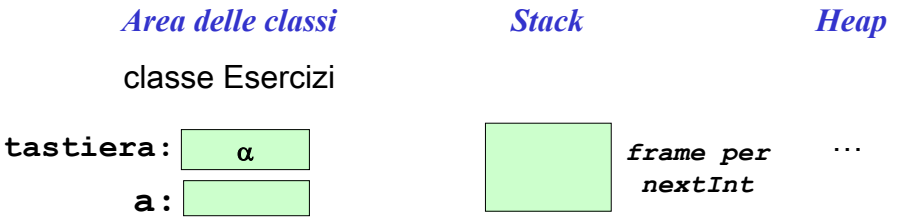
**a:**

α

[Da qui in poi la heap resta cosi']

passo 4

```
class Esercizi{
 static Scanner tastiera=newScanner(System.in)
 static int a = tastiera.nextInt();
```




**l'utente digita 2**



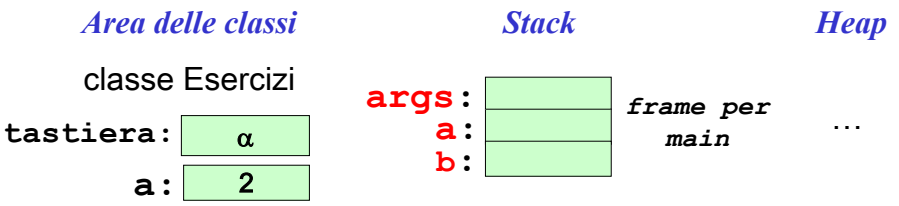
passo 5

```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```


**l'utente ha digitato 2**

## passo 6

```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```

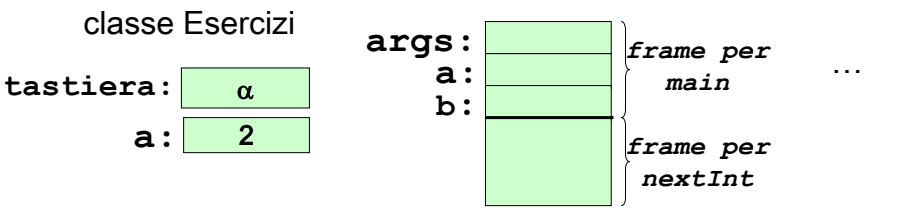



```
  public static void main(String[] args) {
    int a, b;
```



## passo 7

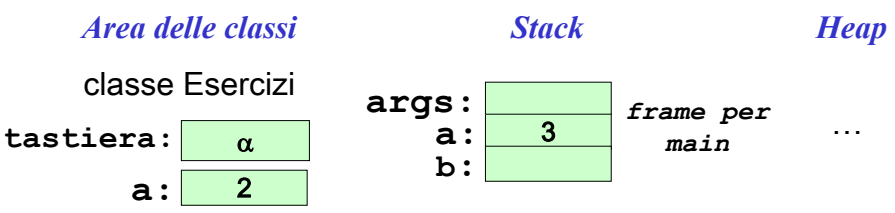
```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```




```
  public static void main(String[] args) {
    int a, b;
    a = tastiera.nextInt();
```

## passo 8

```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```

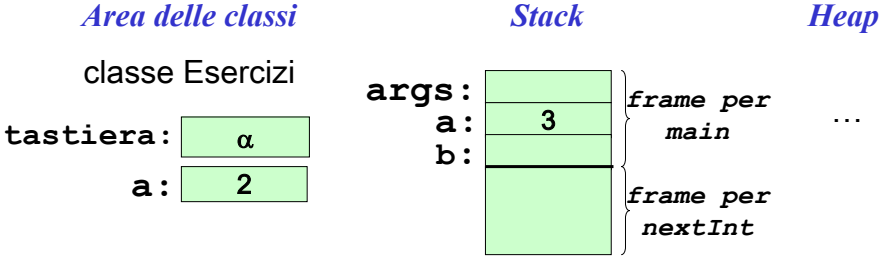



```
  public static void main(String[] args) {
    int a, b;
    a = tastiera.nextInt();
```



## passo 9

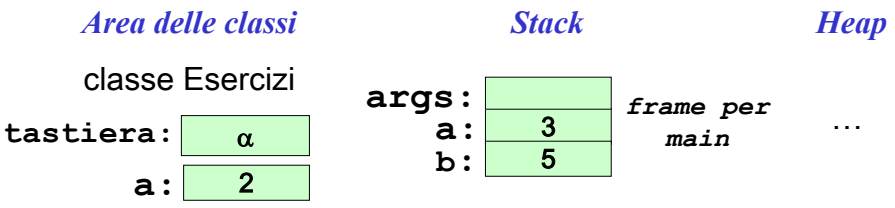
```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```




```
  public static void main(String[] args) {
    int a, b;
    a = tastiera.nextInt();
    b = tastiera.nextInt();
```

## passo 10

```
class Esercizi{
 static Scanner tastiera=newScanner(System.in);
 static int a = tastiera.nextInt();
```

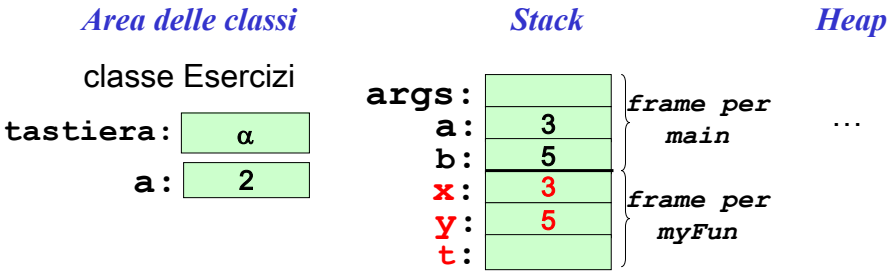



```
  public static void main(String[] args) {
    int a, b;
    a = tastiera.nextInt();
    b = tastiera.nextInt();
```



## passo 11

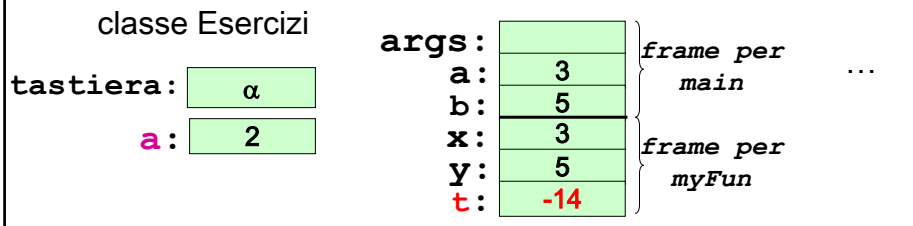
```
class Esercizi{
  …
   static int myFun(int x, int y) {
     int t; ...
```




```
  public static void main(String[] args) {
    ...
   System.out.println(myFun(a,b));
```

passi 13 e 14

```
class Esercizi{
  …
  static int myFun(int x, int y) {
    int t = a*x - 4*y; ...
```

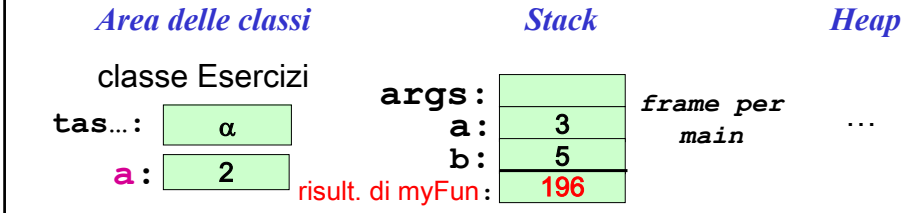



```
  public static void main(String[] args) {
    ...
  System.out.println(myFun(a,b));
```



passo 12

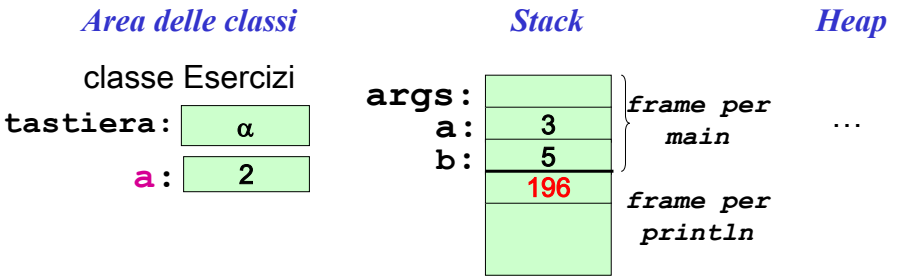
```
class Esercizi{
  …
  static int myFun(int x, int y) {
    int t = a*x - 4*y; return t*t; }
```




```
  public static void main(String[] args) {
    ...
  System.out.println(myFun(a,b));
```

## passo 13

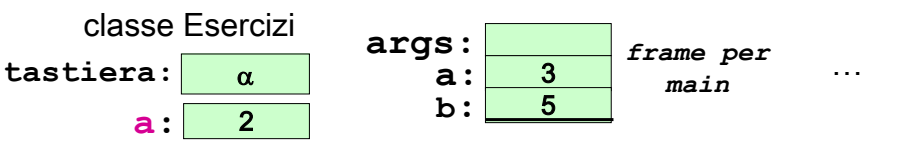
```
class Esercizi{
   ...
   ...
```




```
   public static void main(String[] args) {
      ...
   System.out.println(myFun(a,b));
```



## passo 14



passo 15

Esecuzione del programma terminata