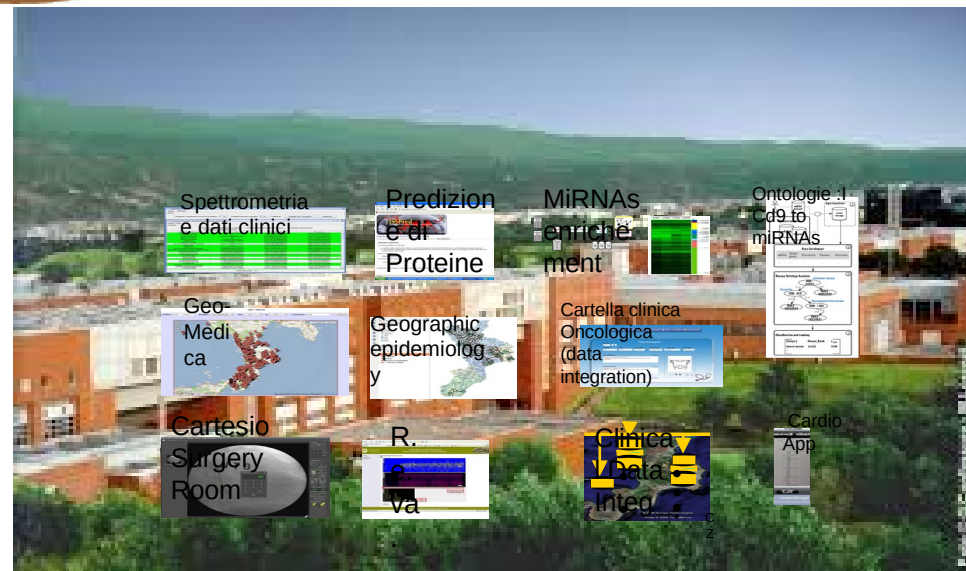


Corso di Tecniche di Programmazione cdI Medicina e Chirurgia TD II anno

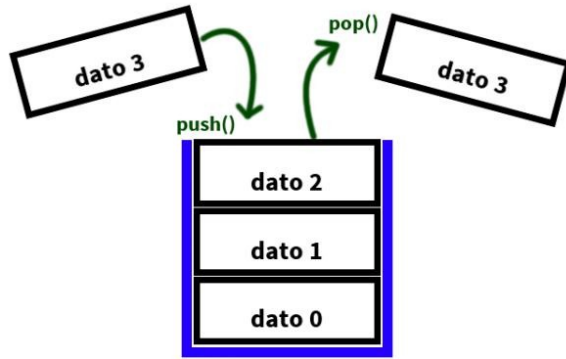


Lezione di oggi



- - Variabili e Record di Attivazione
 - Strutture dati: vettori e matrici
 - Uso di sistemi cloud di compilazione (esempio di replit.com)
 - Introduzione alla ricorsione

Esempi di strutture dati: rappresentazione logica e fisica



Pila (stack): inserimento e cancellazione dallo stesso punto

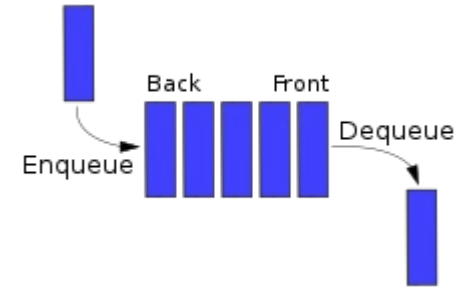
index	0	1	2	3	4	5	6	7	8	9	n-1
	5	1	7	9	5	3	2	9	7	3	

n=dimensione=10

Vettori e Matrici

		Column index					
		[0]	[1]	[2]	[3]	[4]	[5]
Row index	[0]						
	[1]						
	[2]						
	[3]						
	[4]						
	[5]						
	[6]						
	[7]						
	[8]						
	[9]						
	[10]						

balances[3][4]



Coda: inserimento da un lato e rimozione dall'altro

Rappresentazione fisica dei dati



- In memoria si accede sempre con un punto di partenza e un offset
- La gestione della memoria e' lasciata al S.O.
- L'allocazione e deallocazione è lasciata sempre al S.O.
- Alcuni linguaggi di programmazione consentono la gestione della memoria via programma
 - Esempio di caso d'uso: acquisizione di bioimmagini e gestione in memoria: il caso del simulatore (cartesio)

I record

- Il *record* è una struttura dati che può essere eterogenea o omogenea, e quindi può contenere una combinazione di elementi che possono essere di diverso tipo, ad esempio un intero, un numero in virgola mobile e un carattere testuale
- Gli elementi di un record sono detti campi, e sono identificati da un nome
- Ad esempio

Numero complesso

```
struct numero complesso begin  
    float parte reale  
    float parte immaginaria  
end struct
```

I record nella medicina

- Un ulteriore esempio di record – in un contesto medico – è la cartella clinica

Cartella clinica

```
struct cartella clinica begin  
  int numero cartella  
  char  paziente[100]  
  ...  
  struct diagnosi di ingresso begin  
    char  data e ora[20]  
    char  icd9[5]  
    char  note[1000]  
  end struct  
  ...  
end struct
```

Accesso ai contenuti di un record

- L'accesso ai campi di un record avviene usando l'operatore “.”

Accesso ai campi di un record

cartella clinica cc

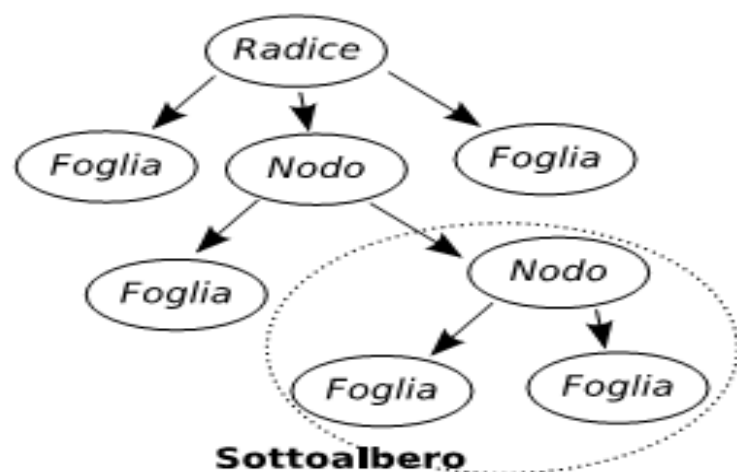
int n ← cc.numero cartella

cc.paziente ← 'Pierpaolo Vittorini'

cc.diagnosi di ingresso.icd9 ← '540.9'

Gli alberi

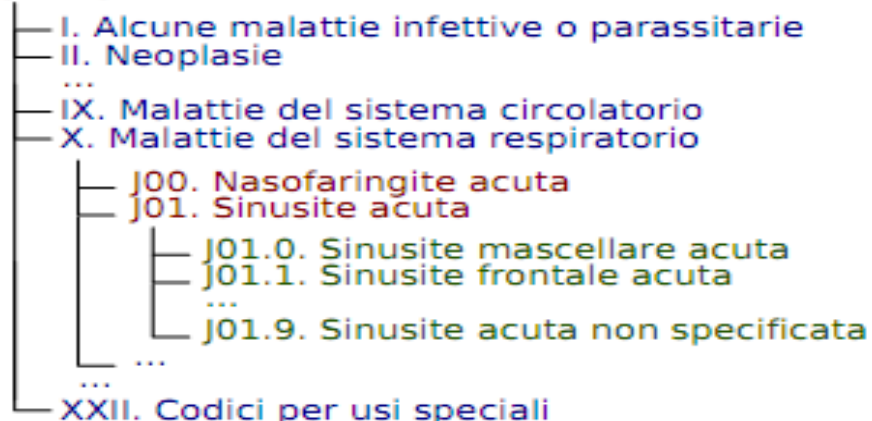
- Un albero è una struttura dati di tipo *gerarchico*
- In un albero, ciascun nodo ha un *padre* e un certo numero di *figli*
- Esiste però un nodo speciale chiamato *radice*, il quale non ha alcun padre
- Esistono altri nodi chiamati *foglie*, i quali non hanno figli
- Una porzione di albero è a sua volta un albero, chiamato *sottoalbero*
- Ciascun nodo porta con sé un'informazione



Gli alberi nella medicina

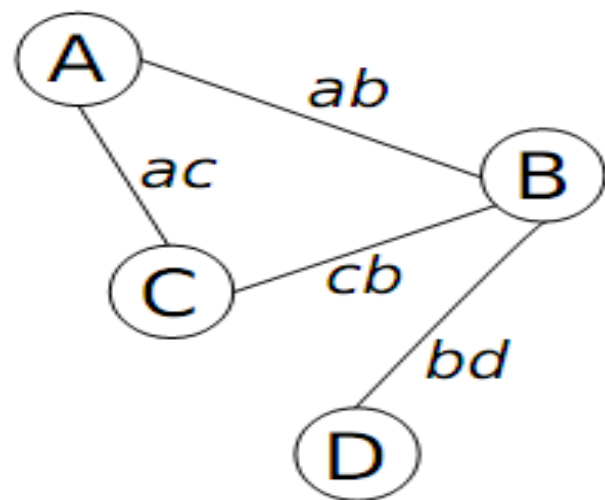
- Un esempio di uso di strutture ad albero nella medicina è relativa alla classificazione delle malattie

ICD10

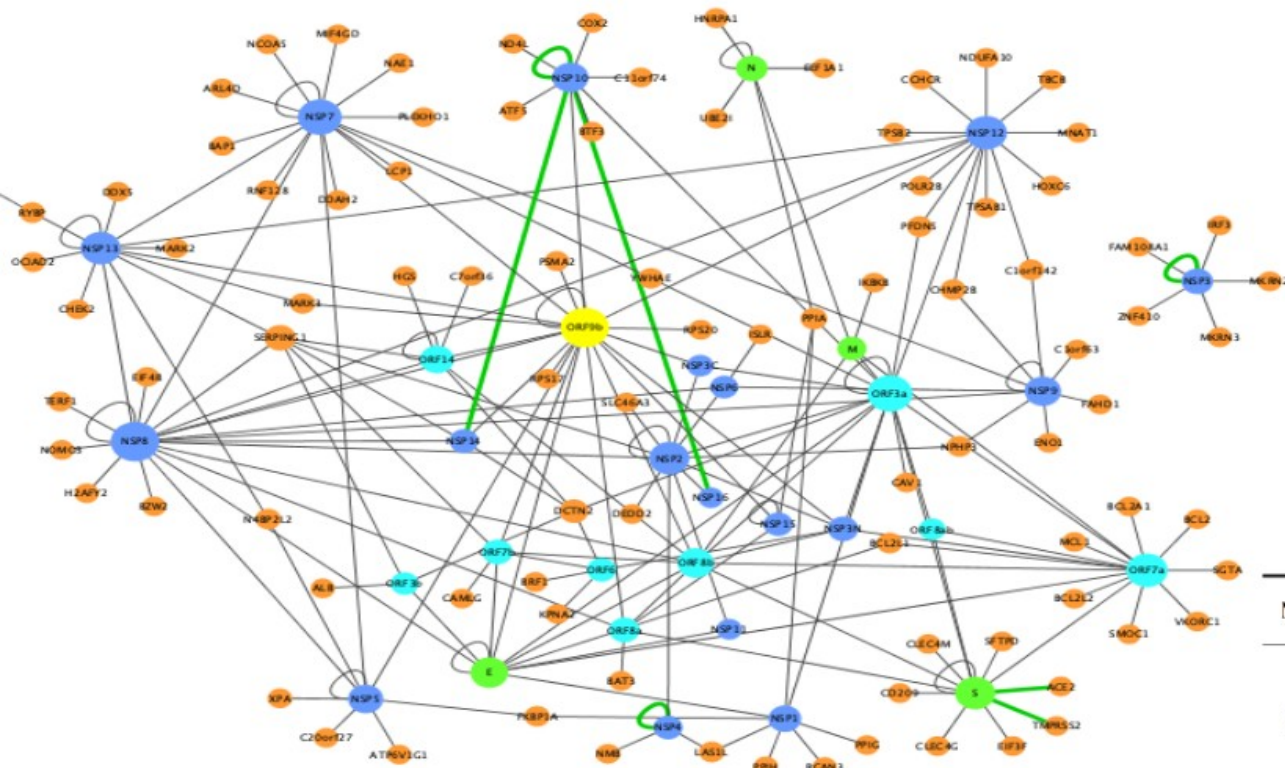
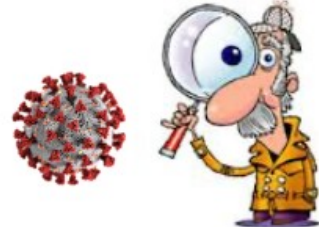




Grafi

- Un *grafo* è un insieme di elementi detti *nodi* o *vertici*, collegati fra loro da *archi* o *lati*, eventualmente dotati di *etichette*
- Il grafo è la struttura dati più complessa e ricca su cui l'informatica può far riferimento



Estrazione di dati Covid-19 e predizione di ICUs



 Human protein
 Covid-19 protein

Network Parameters	SARS-CoV Intra-viral Interactome	SARS-CoV-Host Interactome	Unified Interactome
No. of nodes	31	118	125
No. of edges	86	114	206
No. of components	1	8	2
Diameter	4	14	7
Average degree	4.710	1.95	3.04
Clustering coefficient	0.448	0.0	0.068

Modello a run time per la gestione della memoria



- Il modello run time serve a simulare il comportamento della memoria durante l'esecuzione dei programmi
-



Il **modello runtime** è il modello al tempo di esecuzione dei programmi

- gestione della memoria e della capacità di elaborazione del calcolatore
- allocazione delle aree di memoria
- esecuzione delle operazioni da parte degli oggetti in caso programmazione a oggetti o dell'istanza in corso
- la gestione della memoria avviene per **aree di memoria**
- la gestione della memoria è dinamica
 - basata sulle operazioni di **allocazione** e **deallocazione** di aree di memoria

Esecuzione dei metodi



Il modello di gestione della memoria per l'esecuzione dei metodi (o costruttori) è basato sui **record di attivazione**

- un record di attivazione memorizza le informazioni necessarie a una singola attivazione di un metodo
 - punto di ritorno
 - riferimento all'oggetto esecutore
 - variabili locali
- esecuzione di un metodo
 - all'invocazione, viene allocato un record di attivazione
 - alla terminazione, il record di attivazione viene deallocato

I record di attivazione vengono gestiti mediante una **pila di attivazione**

Esecuzione di Metodi

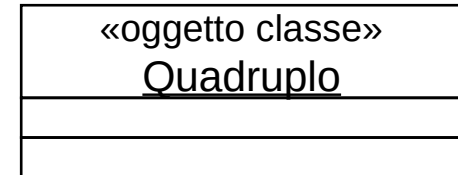


L'applicazione **Quadruplo**

```
class Quadruplo {  
    public static int somma(int a, int b) {  
        int c;  
        c = a+b;  
        return c;  
    }  
    public static int doppio(int n) {  
        int d;  
        d = somma(n,n);  
        return d;  
    }  
    public static void main(String[] args) {  
        int a, b, c;  
        a = 2;  
        b = somma(a,a);  
        c = doppio(b);  
    }  
}
```

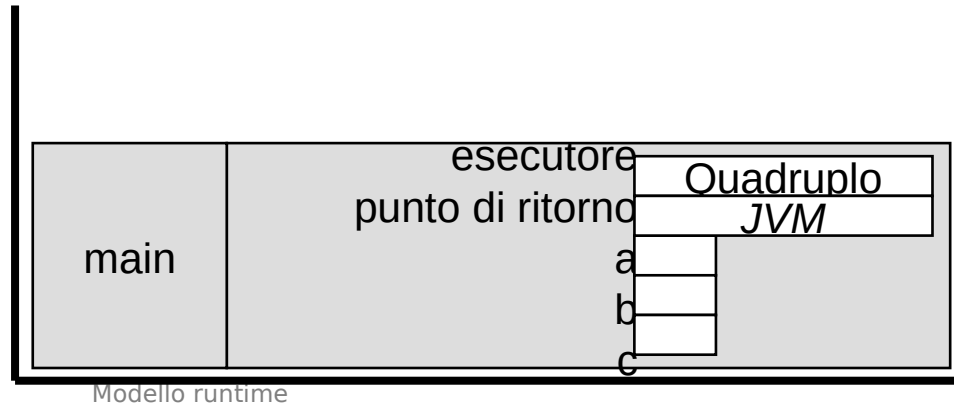

t=0 — avvio dell'esecuzione Quadruplo

L'esecuzione dell'applicazione **Quadruplo** viene richiesta mediante l'esecuzione del comando **java Quadruplo**



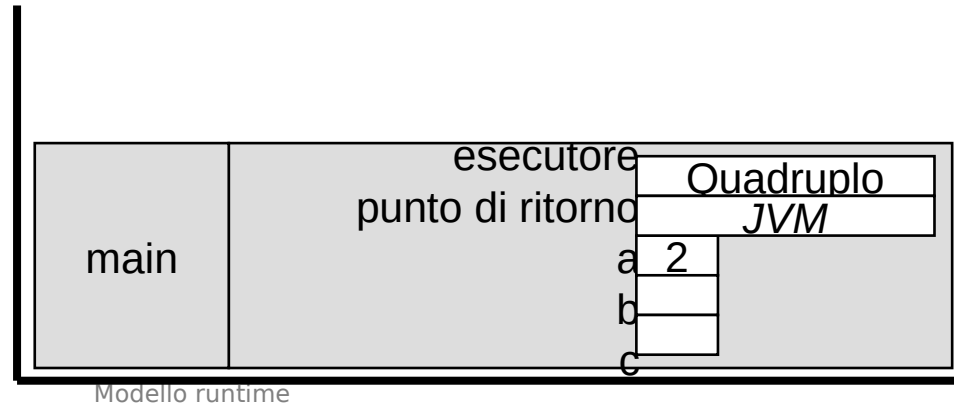
t=1 — attivazione di main

La JVM chiede all'oggetto **Quadruplo** di eseguire il metodo **main**



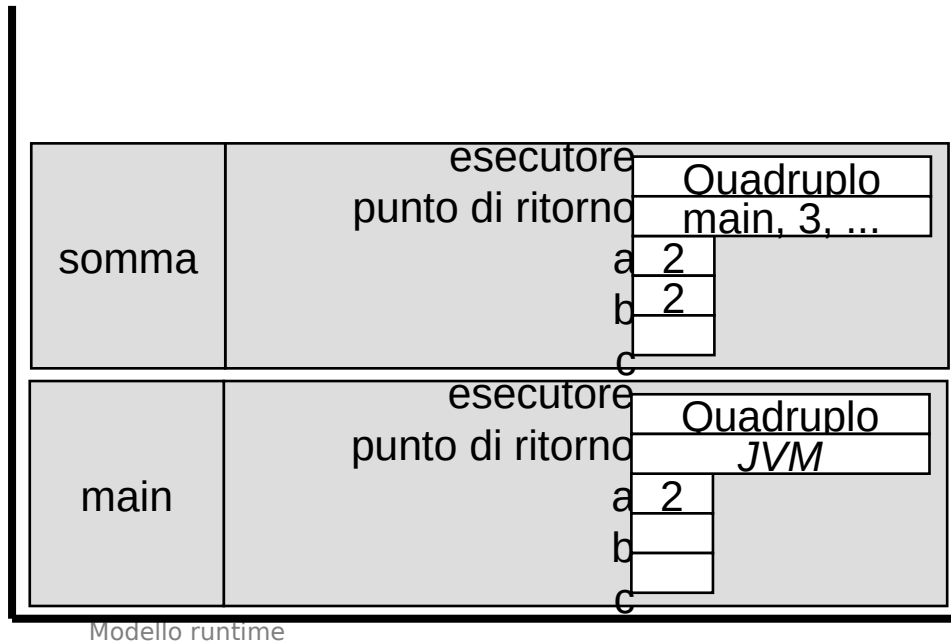
t=2 — assegnazione a una variabile

Viene eseguita l'assegnazione **a=2**



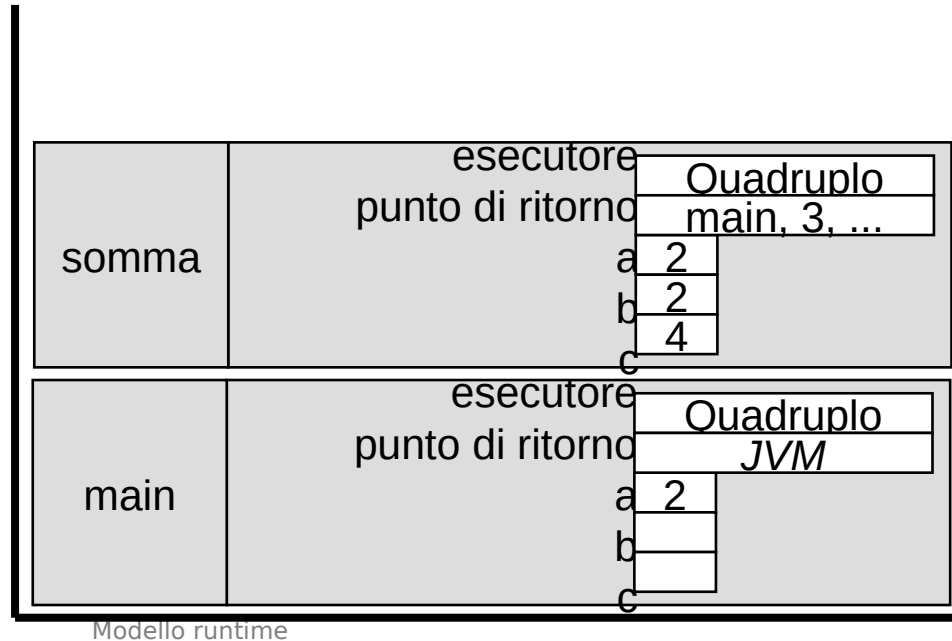
t=3 — invocazione e attivazione di somma

Il metodo **main** invoca il metodo **somma**



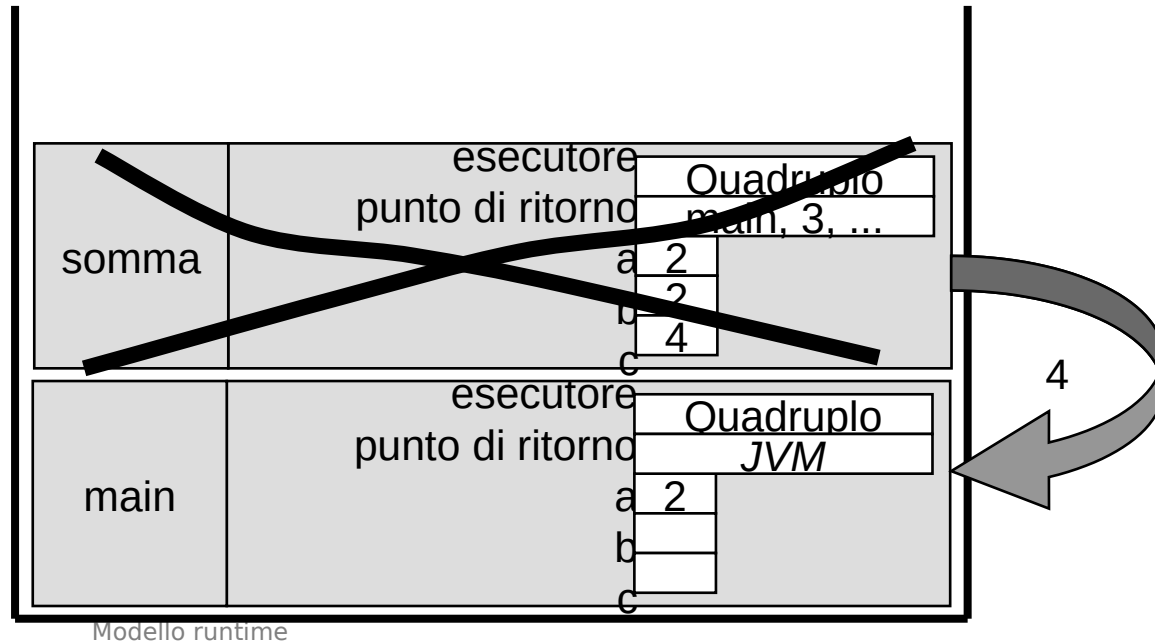
t=4 — esecuzione di una assegnazione

Viene eseguita l'assegnazione **c=a+b**



t=5 — terminazione di somma

Viene eseguita l'istruzione **return c** di **somma**

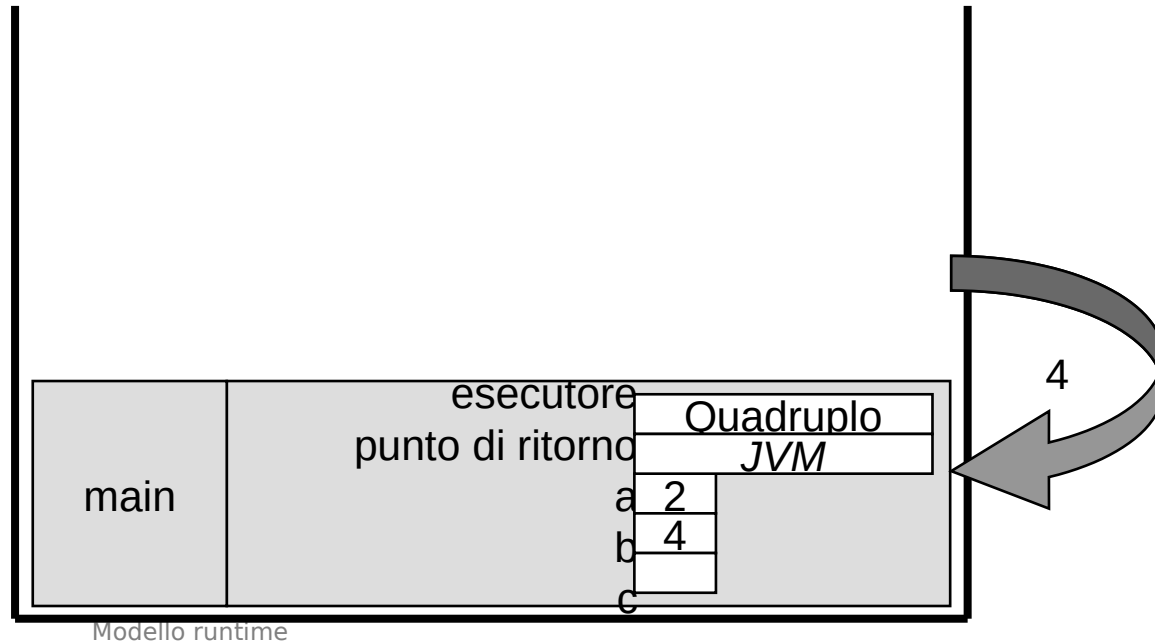


L'applicazione **Quadruplo**

```
class Quadruplo {  
    public static int somma(int a, int b) {  
        int c;  
        c = a+b;  
        return c;  
    }  
    public static int doppio(int n) {  
        int d;  
        d = somma(n,n);  
        return d;  
    }  
    public static void main(String[] args) {  
        int a, b, c;  
        a = 2;  
        b = somma(a,a);  
        c = doppio(b);  
    }  
}
```

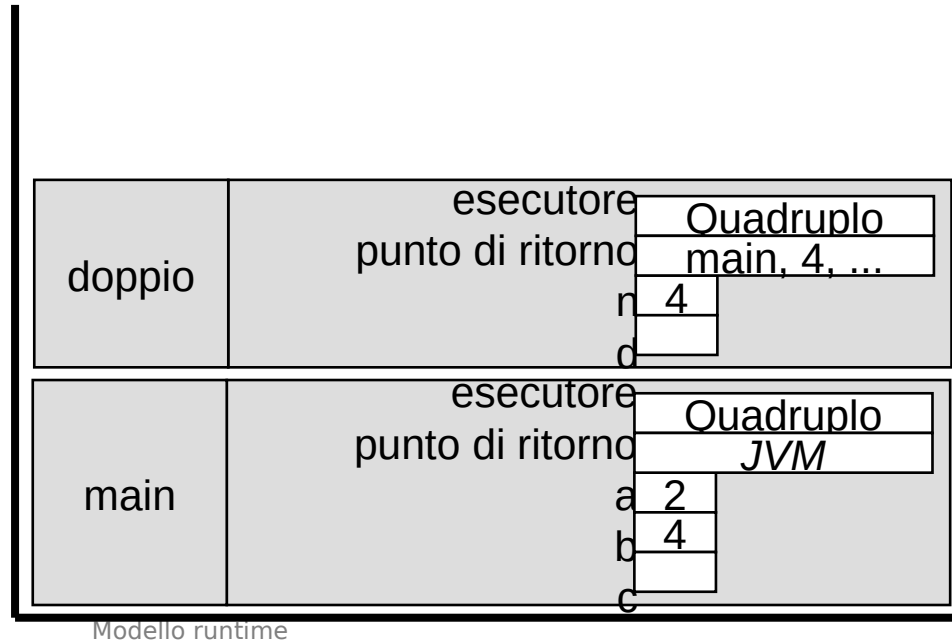
t=6 — assegnazione del valore restituito

Viene completata l'assegnazione alla variabile **b**



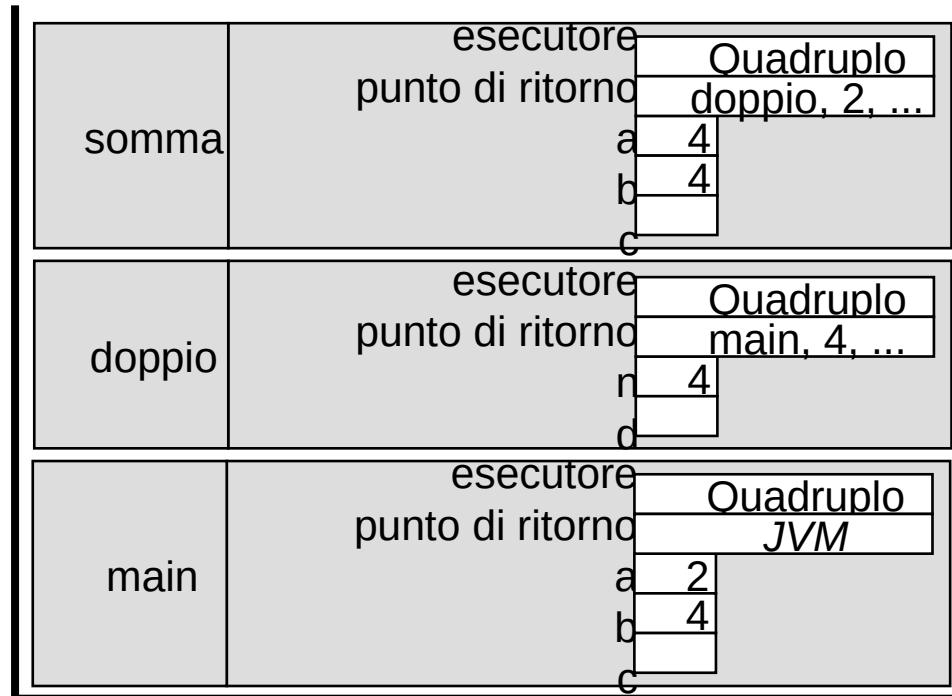
t=7 — invocazione e attivazione di doppio

Il metodo **main** invoca il metodo **doppio**



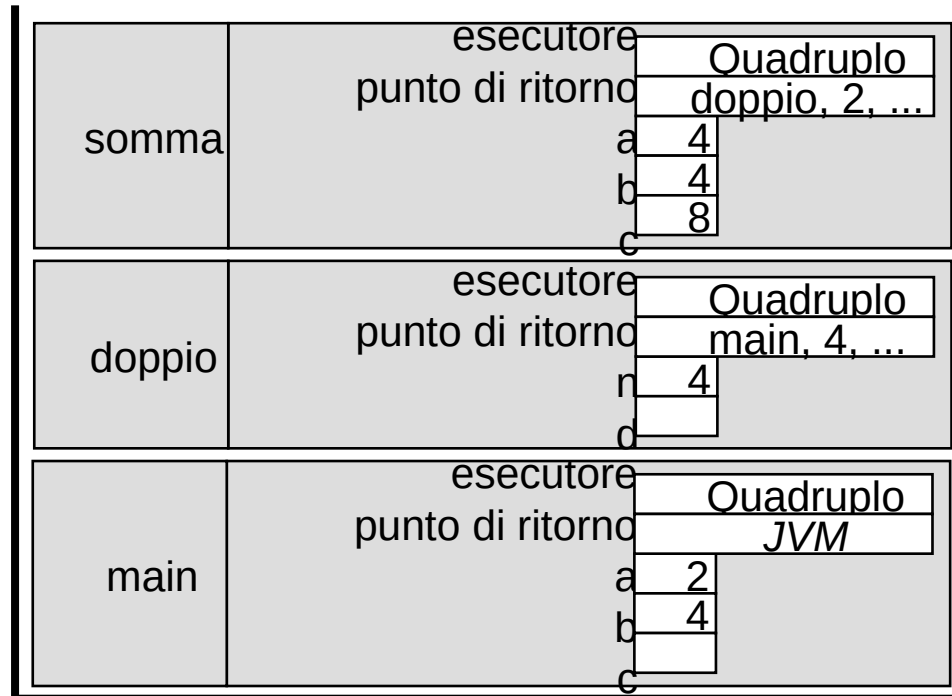
t=8 — invocazione e attivazione di somma

Il metodo **doppio** invoca il metodo **somma**



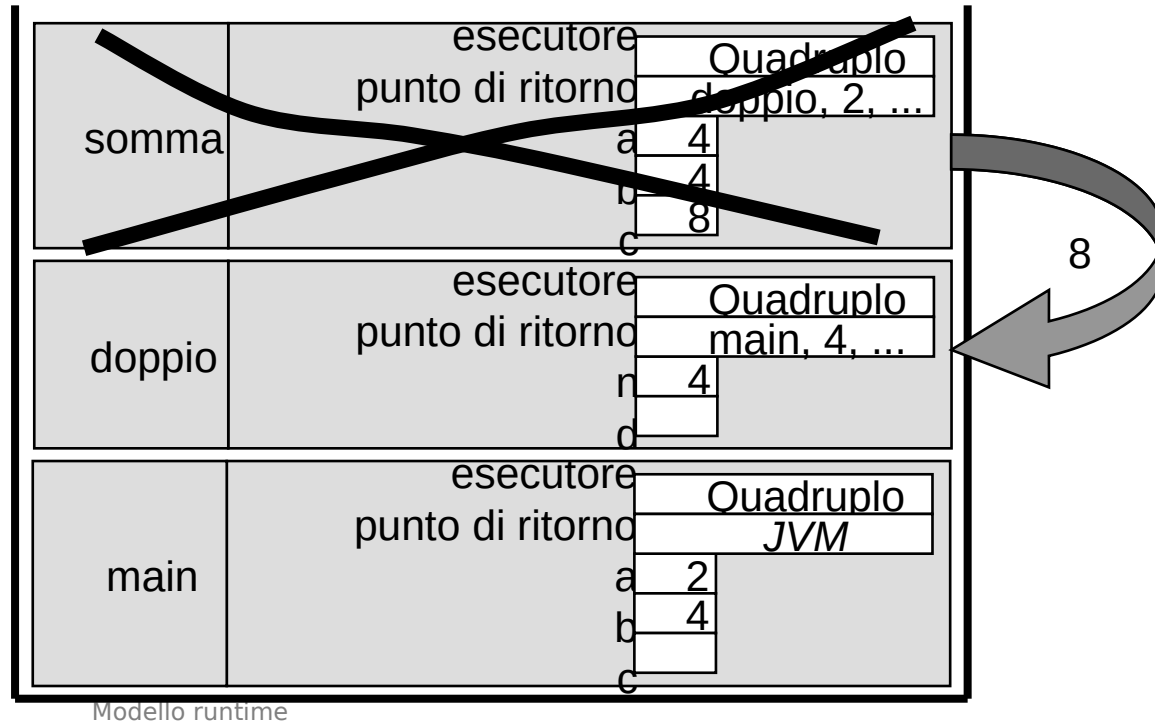
t=9 — esecuzione di una assegnazione

Viene eseguita l'assegnazione **c=a+b**



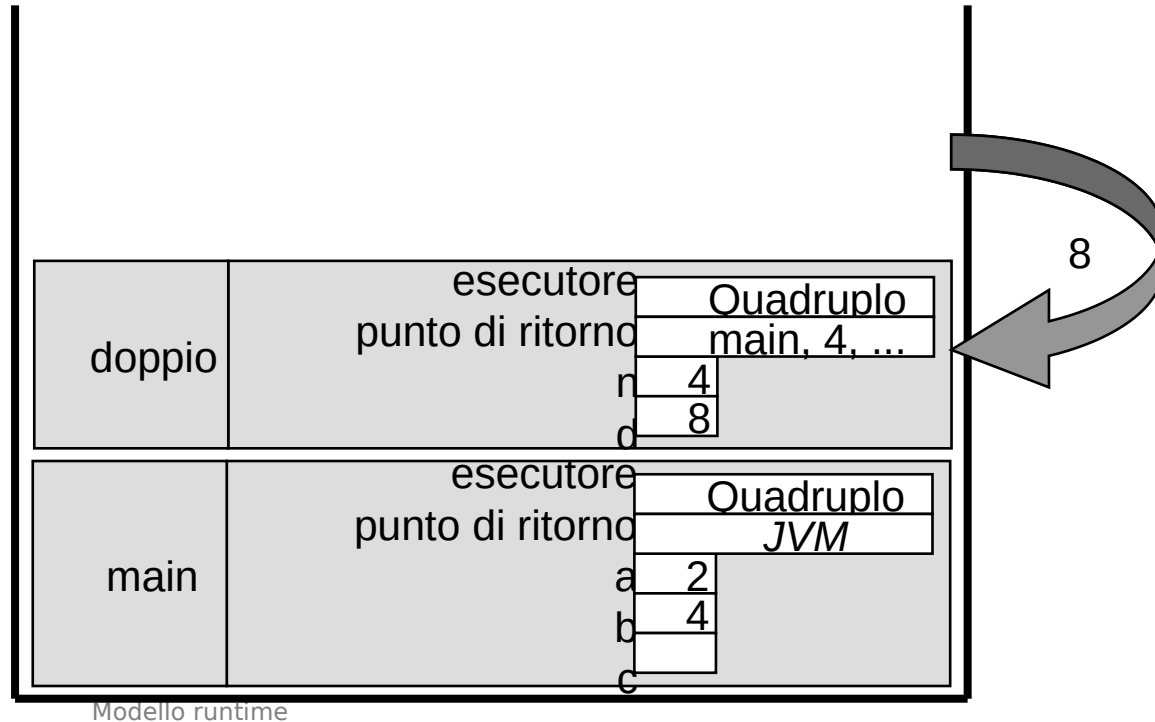
t=10 — terminazione di somma

Viene eseguita l'istruzione **return c** di **somma**



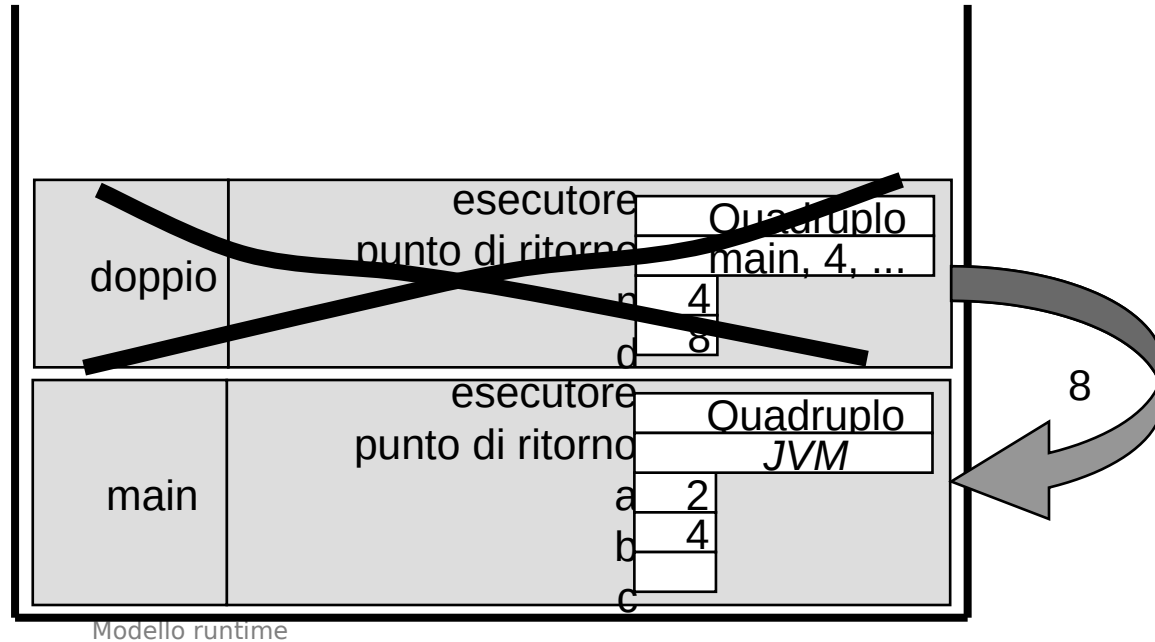
t=11 — assegnazione del valore restituito

Viene completata l'assegnazione alla variabile **d**



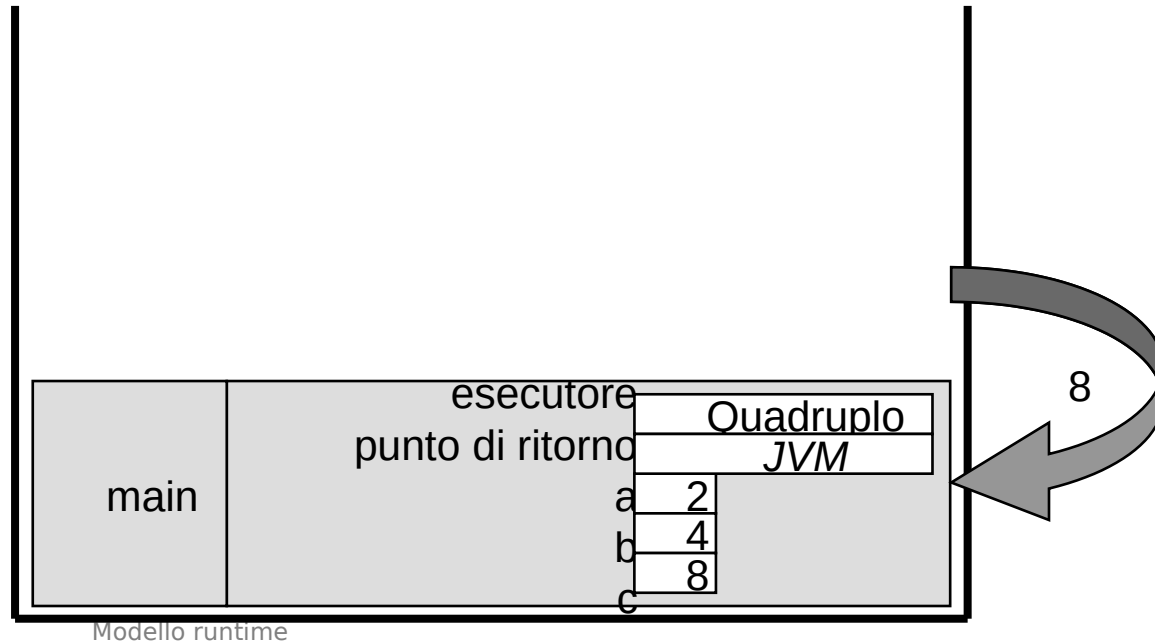
t=12 — terminazione di doppio

Viene eseguita l'istruzione **return d** di **doppio**



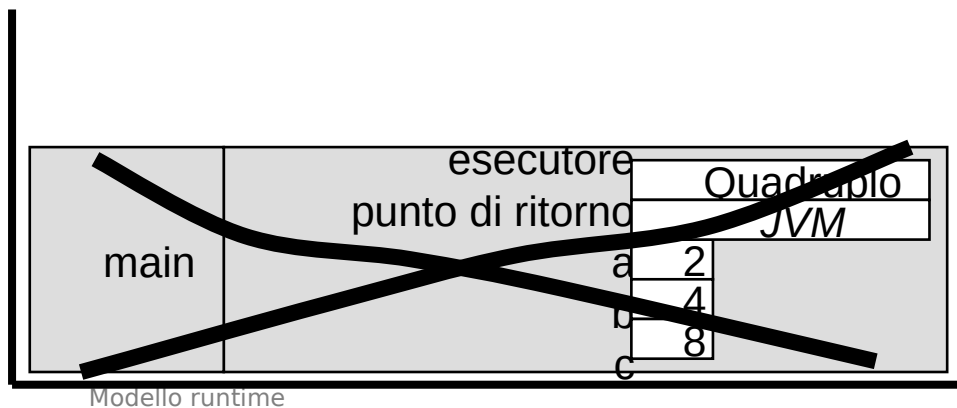
t=13 — assegnazione del valore restituito

Viene completata l'assegnazione alla variabile **c**



t=14 — terminazione di main

Termina anche l'esecuzione del metodo **main**

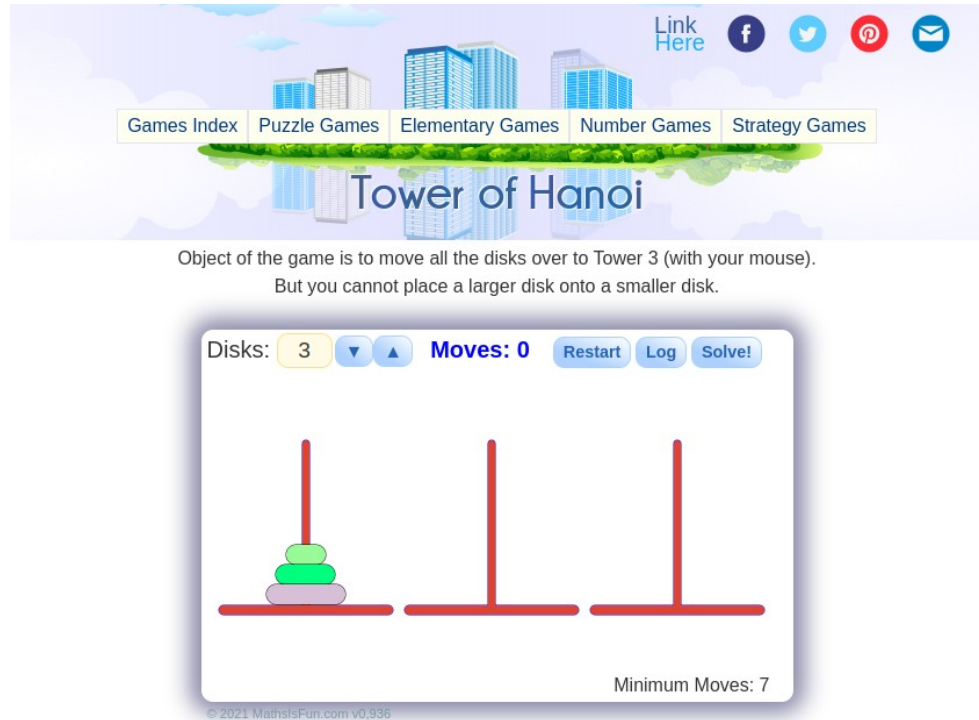


Esecuzione di metodi: discussione

Osservazioni

- i record di attivazione sono relativi alle attivazioni dei metodi
- ordine nell'allocazione/deallocazione di record di attivazione
- in questo esempio è possibile pensare a una gestione statica della memoria
 - in generale non è possibile

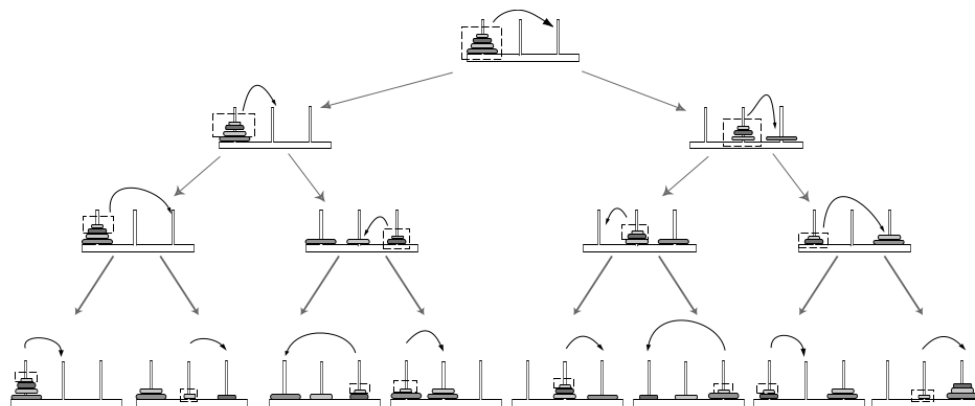
Esempio di ricorsione: La torre di Hanoi



Lo scopo del gioco è portare tutti i dischi su un paletto diverso, potendo spostare solo un disco alla volta e potendo mettere un disco solo su un altro disco più grande, mai su uno più piccolo.

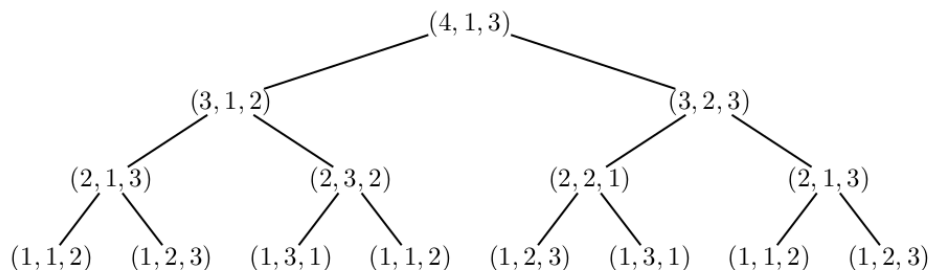
Link da [wikipedia](#):

<https://www.mathsisfun.com/games/towerofhanoi.html>



Il problema si risolve in modo semplice
In termini di linee di codice in modo
Ricorsivo (vedremo)

“To solve a five-disk tower requires 31 moves, but to solve a hundred-disk tower would require more moves than there are atoms in the universe.”



```
HANOITOWERS(n, fromPeg, toPeg)
1  if n = 1
2      output "Move disk from peg fromPeg to peg toPeg"
3      return
4  unusedPeg ← 6 - fromPeg - toPeg
5  HANOITOWERS(n - 1, fromPeg, unusedPeg)
6  output "Move disk from peg fromPeg to peg toPeg"
7  HANOITOWERS(n - 1, unusedPeg, toPeg)
8  return
```

Esempio di variabili e di controllo



- <https://replit.com/@PierangeloV/TestLearning#main.py>

```
var_num=3;
```

```
if(var_num==3):
```

```
    print("uguale 3")
```

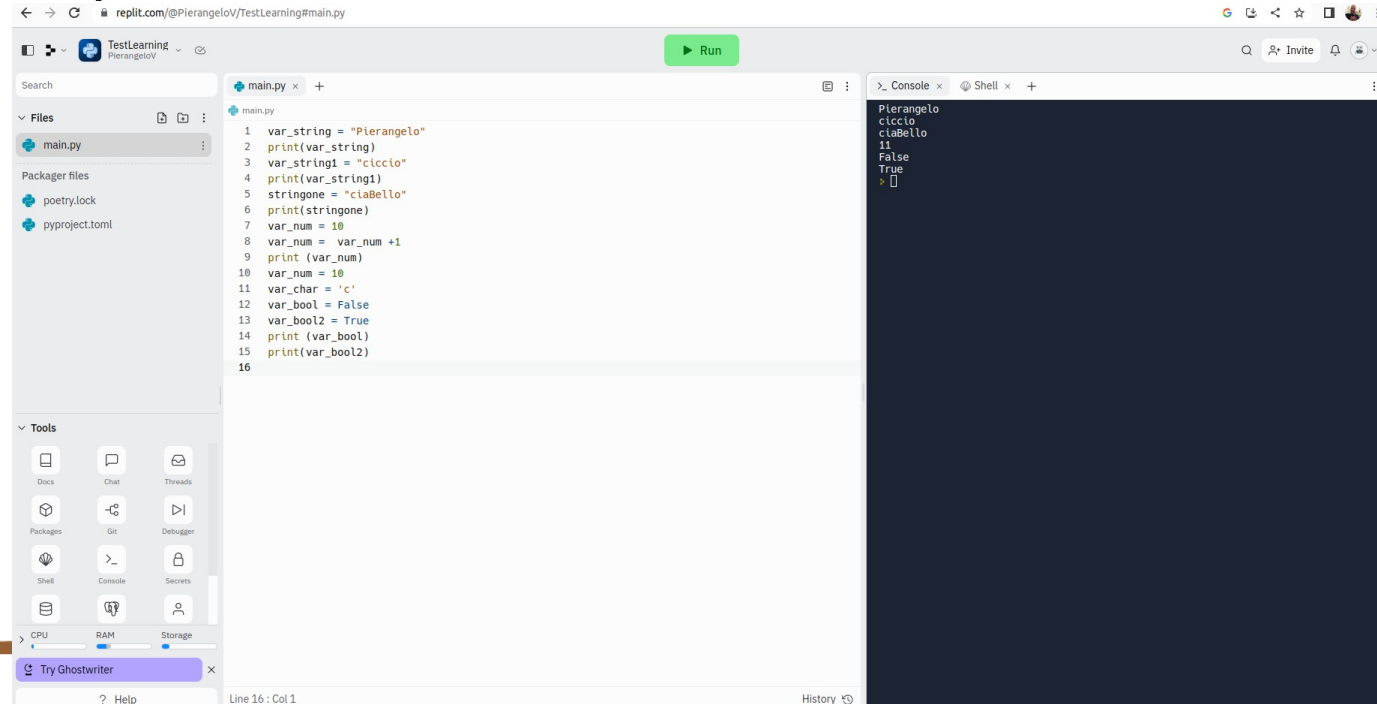
```
else:
```

```
    print("non e' uguale 3")
```

- Concetto di variabile e di controllo. Riferimenti alle strutture a blocchi
- Concetto di tipi di dati meno rigoroso da linguaggi standard



- Sistemi cloud based per la compilazione e definizione di spazio di lavoro e di coding
- Senza installare compilatori
- <https://replit.com>
-



Come definire un esempio per lavorare in modo semplice



- Un linguaggio di programmazione ha bisogno di
 - Sintassi, semantica, un compilatore, un traduttore e un esecutore
- Definire uno spazio di lavoro in cloud: un esempio
 - Replit.com; definire un account con un nome e una mail
 - Avviare un nuovo spazio di lavoro,
 - Scegliere l'ambiente Python
 - Scrivere il codice nel main.py
 - Se si scrive un altro file, definire `import Nomefile.py` nel main



```
24 var_num= 3
25 var_num-=2
26 ✓ if (var_num<=5 and var_num>2):
27     print("Variabile minore uguale di 5 e maggiore 2")
28 ✓ elif (var_num>2):
29     printf("Var magg 2")
30 ✓ else:
31     print("fine")
```

Ancora richiami (rapidi) di Python



- Nel main.py si puo' usare "import esMatrici"

esMatrici.py

```
1  vettore_v = [1,12,-23]
2  palazzo_v = ["mario", "luca", "giovanni", "andrea"]
3  matrice_m = [[12, 11, 10],[-1, 22, 30 ], [7,12,15]]
4  # m e' una matrice
5  # quadrata
6  print(vettore_v[0])
7  #scrivo il valore nelal prima posizione
8  print(palazzo_v[1])
9  print (matrice_m[0][1])
```


Cicli



- Sempre in reply.it. Il for esegue n con n il numero di valori che stanno nel vettore

```
1 counter = 0
2 while (counter<5):
3     print("Pippo", counter)
4     counter=counter+1
5 print ("ora il ciclo con il for")
6 vettore_v = [12,13,1,7,2]
7 for n in vettore_v:
8     print(n)
9 for n in vettore_v:
10     print("sei bravo")
11
```

```
la variabile beta e Pterangeto
fine
1
luca
11
Pippo 0
Pippo 1
Pippo 2
Pippo 3
Pippo 4
ora il ciclo con il for
12
13
1
7
2
sei bravo
sei bravo
sei bravo
sei bravo
sei bravo
sei bravo
❖
```

Shell x +

Chiamate a procedure



- `def function_f1(m,i,j):`
- `print (m[i],[j])`
- `matrice_m = [1,2,85,5],[6,7,8,9]`
- `function_f1(matrice_m,1,3)`
- `def funzione (a,b):`
- `return a+b`
- `result = funzione (3,5)`
- `print (result)`

The screenshot shows a Python IDE with three tabs: Procedure.py, main.py, and esMatrici.py. The Procedure.py tab is active, displaying the following code:

```
1 def procedure_p():
2     print ("ciao")
3     print("bello")
4 procedure_p()
5 procedure_p()
6 def function_f(a,b):
7     print(a+b)
8 function_f(4,5)
9 def function_f1(m,i,j):
10     print (m[i],[j])
11 matrice_m = [1,2,85,5],[6,7,8,9]
12 function_f1(matrice_m,1,3)
13 def funzione (a,b):
14     return a+b
15 result = funzione (3,5)
16 print (result)
17
```

The output console on the right shows the following text:

```
11
Pippo 0
Pippo 1
Pippo 2
Pippo 3
Pippo 4
ora il ciclo con i
12
13
1
7
2
sei bravo
sei bravo
sei bravo
sei bravo
ciao
bello
ciao
bello
9
[6, 7, 8, 9] [3]
8
> []
```

At the bottom, a shell window shows the command prompt:

```
~/TestLearning$
```

Classi ed esempi di programmazione a oggetti



```
class Persona:
    def __init__(self, a, b, c):
        self.nome = a
        self.cognome = b
        self.numeroAsn = c

p1 = Persona("Pierangelo", "Veltri", 123456789)
p2 = Persona("Luca", "Rossi", 102304050606)
print (p1.nome, p1.cognome, p1.numeroAsn)
print (p2.nome, p2.cognome, p2.numeroAsn)
```