

Considera il seguente codice:

```
1  public class StackHeapSimulation {
2
3      public static void main(String[] args) {
4          // Dichiarazione di variabili locali
5          int localVar = 20;
6
7          // Creazione di un oggetto
8          MyObject myObj = new MyObject(localVar);
9
10         // Passaggio di parametri
11         modifyValue(localVar, myObj);
12
13         // Richiamo del metodo updateValue() su myObj
14         myObj.updateValue(50);
15     }
16
17     public static void modifyValue(int value, MyObject obj) {
18         value = 30;
19         obj.value = value; // Modifica il valore dell'oggetto
20     }
21 }
22
23 // Classe MyObject
24 class MyObject {
25     int value;
26
27     public MyObject(int value) {
28         this.value = value;
29     }
30
31     // Metodo che aggiorna il valore dell'oggetto
32     void updateValue(int newValue) {
33         this.value = newValue;
34     }
35 }
```

simula l'evoluzione di stack e heap durante l'esecuzione. **Traccia lo stack e l'heap** manualmente:

- Disegna lo **stack** per ogni funzione chiamata, indicando:
 - Parametri formali.
 - Variabili locali.
 - Indirizzo di ritorno.
- Disegna l'**heap** per rappresentare la memoria allocata dinamicamente (oggetti MyObject).
- Mostra chiaramente come lo stack cresce e si riduce durante l'esecuzione, e come gli oggetti nell'heap vengono modificati.

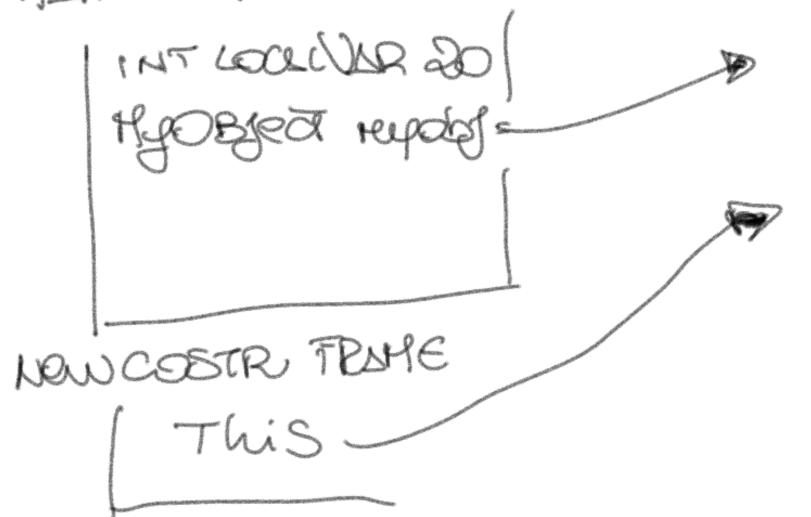
Visualizza chiaramente:

- Come lo **stack cresce** con ogni nuova chiamata a funzione e si **riduce** al ritorno.
- Come gli oggetti nell'**heap vengono modificati** durante l'esecuzione (ad esempio, quando vengono modificati i campi dell'oggetto MyObject).

Verifica con lo strumento online come [Python Tutor](#) che la tua simulazione sia corretta.

1STR R4D 5-8 → push ~~HyObj~~
push constructor
STACK

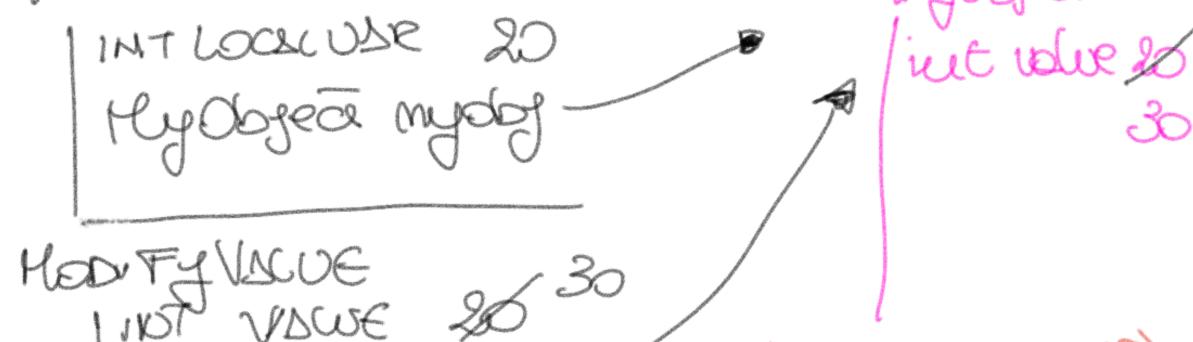
MAIN FRAME



1 STR R4D 9 -10 pop constructor
push modify Value
(stloc)

Stack

MAIN FRAME



NON ^w
VAR
Locale

Ho due Possibili forme

INT VALUE e MyObject obj

Ricche 12 - 13 pop Metodi Statici

modify value

push updateValue

resue value resuse
stato

f

(stack)

MAIN FRAME

INT locValue 20

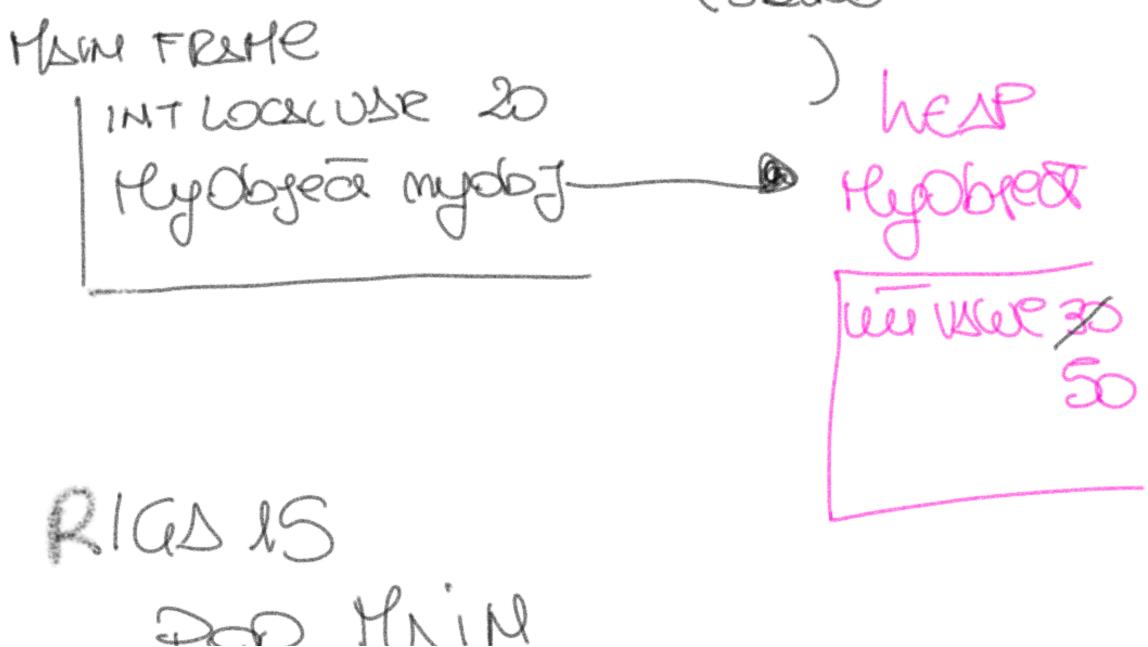
MyObject myobj

updateValue FRAME

heap

MyObject

locValue 30
50



In seguito, rispondi alle seguenti domande:

1. Qual è la differenza tra la variabile localVar e l'oggetto myObj in termini di allocazione della memoria?

- Spiega come localVar è allocato nello stack, mentre myObj è allocato dinamicamente nell'heap.
- **Risposta** localVar è una variabile primitiva (un intero) e viene allocata nello **stack**. Ogni volta che si chiama una funzione in Java (in questo caso main), le variabili locali come localVar vengono allocate in una porzione dello stack.
- **Risposta** myObj è un oggetto che viene allocato nell'**heap**, la porzione di memoria usata per l'allocazione dinamica. Il puntatore (riferimento all'oggetto) a myObj viene memorizzato nello stack, ma l'oggetto stesso vive nell'heap, dove la sua memoria può crescere e cambiare durante l'esecuzione del programma.

2. Cosa è successo al valore di localVar all'interno del metodo modifyValue ? Perché?

- Spiega perché la modifica di localVar all'interno del metodo non persiste al di fuori del metodo.

Risposta Quando localVar viene passato a modifyValue (metodo statico), viene **passato per valore**, cioè viene fatta una copia di localVar. All'interno di modifyValue, il parametro value (che contiene la copia) viene modificato a 30, ma questo non influisce su localVar fuori dalla funzione perché stiamo operando su una copia, non sulla variabile originale. Una volta che la funzione termina, la copia scompare e localVar rimane con il suo valore iniziale di 20.

3. Perché l'oggetto myObj ha mantenuto il suo valore anche dopo la modifica all'interno del metodo?

- Descrivi il comportamento dell'heap e come la modifica del campo value di myObj ha effetti al di fuori della funzione. L'oggetto myObj viene **passato per riferimento**. Questo significa che, quando myObj viene passato alla funzione modifyValue, non viene creata una copia dell'oggetto, ma viene passato il riferimento (indirizzo di memoria) all'oggetto effettivo nell'heap. Di conseguenza, le modifiche al campo value di myObj all'interno di modifyValue influiscono sull'oggetto stesso, e queste modifiche sono visibili anche fuori dal metodo. In modifyValue, il campo value viene cambiato a 30 e poi a 50, e questa modifica persistono.

4. Perché localVar non è stato modificato all'interno del metodo modifyValue , mentre myObj.value è stato aggiornato?

- Chiarisci la differenza tra il passaggio di parametri per valore (per i tipi primitivi come localVar) e per riferimento (per gli oggetti come myObj).

Risposta localVar è un tipo **primitivo** e, come tale, viene passato per **valore**. Questo significa che viene fatta una copia del valore di localVar e le modifiche avvengono sulla copia. Quindi, la modifica non influisce sul valore di localVar originale. Al contrario, gli oggetti in Java vengono passati per **riferimento**, quindi myObj non viene copiato. Al suo posto viene passato un riferimento all'oggetto originale nell'heap. Qualsiasi modifica fatta a myObj.value all'interno del metodo modifica effettivamente l'oggetto stesso, e quindi la modifica persiste anche dopo l'uscita dalla funzione.

5. Cosa accade nello stack e nell'heap quando viene chiamato updateValue? Dove è memorizzato il valore modificato?

- Descrivi come la chiamata a updateValue modifica il valore nell'heap e l'impatto sull'oggetto myObj.

Risposta Quando updateValue viene chiamato su myObj, viene creato un nuovo frame nello **stack** per gestire la chiamata a updateValue. Questo frame contiene un riferimento a myObj (il campo this) e il parametro newValue

passato alla funzione. Notare che il metodo statico modifyValue non ha il campo this ma il riferimento all'oggetto è memorizzato nel parametro formale obj. I metodi statici sono legati alla classe, i metodi di istanza sono legati a una specifica istanza della classe, ovvero un oggetto.

6. **Dopo la chiamata al metodo updateValue , quali sono le modifiche nell'heap? Dove viene memorizzato il nuovo valore?**

- Dettaglia come e dove nell'heap viene memorizzato il nuovo valore aggiornato dell'oggetto myObj.

risposta Dopo la chiamata al metodo updateValue, il campo value dell'oggetto myObj nell'**heap** è stato aggiornato al nuovo valore di 50. Questo significa che l'heap contiene ancora lo stesso oggetto myObj, ma il suo campo value ora ha il valore aggiornato. La modifica è memorizzata direttamente nella porzione di memoria heap dove l'oggetto myObj è stato allocato.