Considera il seguente codice:

```java
public class StackHeapSimulation {

    public static void main(String[] args) {
        // Dichiarazione di variabili locali
        int localVar = 20;

        // Creazione di un oggetto
        MyObject myObj = new MyObject(localVar);

        // Passaggio di parametri
        modifyValue(localVar, myObj);

        // Richiamo del metodo updateValue() su myObj
        myObj.updateValue(50);
    }

    public static void modifyValue(int value, MyObject obj) {
        value = 30;
        obj.value = value; // Modifica il valore dell'oggetto
    }
}

// Classe MyObject
class MyObject {
    int value;

    public MyObject(int value) {
        this.value = value;
    }

    // Metodo che aggiorna il valore dell'oggetto
    void updateValue(int newValue) {
        this.value = newValue;
    }
}
```

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.

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.
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.

3. 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.
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).
5. **Cosa accade nello stack e nell’heap quando viene chiamato il metodo** updateValue **? Dove è memorizzato il valore modificato?**
   • Descrivi come la chiamata a updateValue modifica il valore nell’heap e l’impatto sull’oggetto myObj.
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.