Print a Reversed Linked List

Given a linked list, access each node in such a way so that the linked list will print in a reversed manner. You may not change the content of the list.

The following illustration explains this concept:


```
class LinkedList {
  // Linked list Node
  static class Node {
    int value;
    Node next;
  }

  public static void reverse(Node head) {

    // Check case
    if (head == null) {
      return;
    }

    // Recursive case
    else {
      reverse(head.next);
      System.out.print(head.value + " ");
    }
  }

  static Node insertAtHead(Node temp_head, int new_value) {
    Node new_node = new Node();
    new_node.value = new_value;
    new_node.next = (temp_head);
    (temp_head) = new_node;
```

Run

Save Reset

Understanding the Code

The code given above can be broken down into **two parts**. The **recursive method** and the **main** where the method is called.

Driver Method

The driver code is found from **line 33 to line 53**.

- In the driver code, between **lines 37 and 41**, a linked list is created by inserting five nodes using the `insertAtHead` method.
- The `reverse` method is called on **line 52**, which takes only 1 argument:- the `head` of the list.

Recursive Method

Every recursive method consists of two parts: the **base case** and the **recursive case**.

Base Case

The base case is defined on **line 12**. If the `head` of the linked list is null, it indicates that the entire list has been traversed, and the method terminates. This also applies when the list is empty and the method should simply terminate.

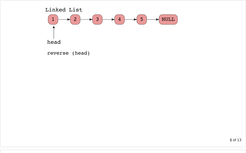
Recursive Case

The recursive case is defined on **line 18**.

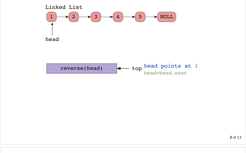
- The method takes only 1 argument, the `head` of the list, which points to the first node of the list. The `head` of the list gets updated in each call.
- Initially, the `head` points to the start node. In every successive recursive call, the `head` points to the next node.
- When the `head` points to null, meaning it reaches the end of the list, the base case is reached, and it prints the values of the node in a reverse manner.

Understanding Through a Stack

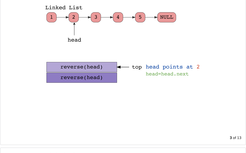
The following illustration helps to explains the code through a stack:



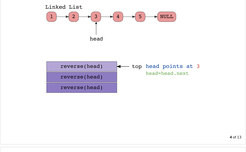
1 of 13



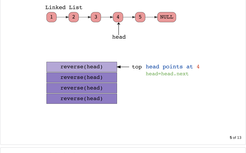
2 of 13



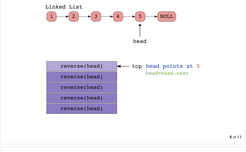
3 of 13



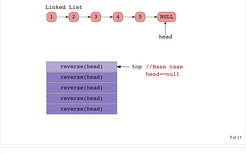
4 of 13



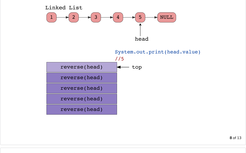
5 of 13



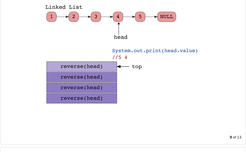
6 of 13



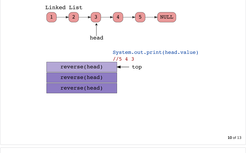
7 of 13



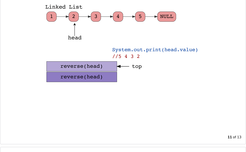
8 of 13



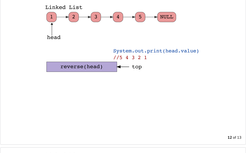
9 of 13



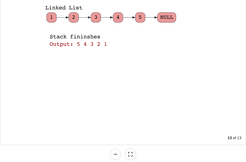
10 of 13



11 of 13



12 of 13



13 of 13