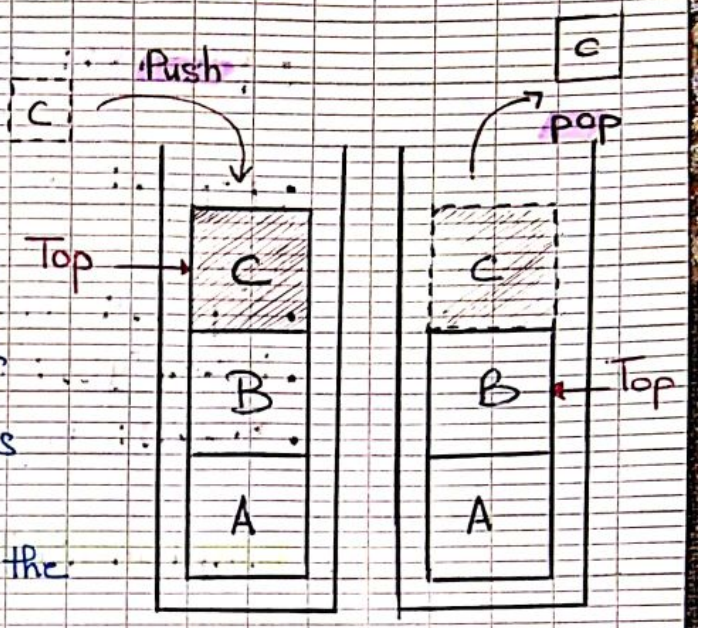
Stack

What is stack



A stack is a linear data structure in which the insertion of a new element and removal of an existing element takes place at the same end represented as the top of the stack.

To implement the stack, it is required to maintain the pointer to the Top of the stack, which is the last element to be inserted because we can access the elements only on the top of the stack

LIFO (Last In First Out)

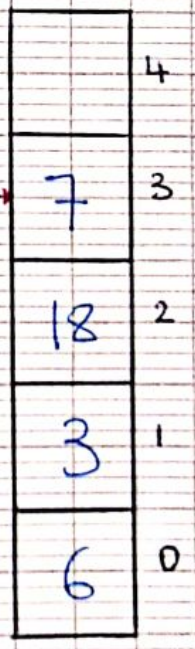
This strategy states that the last element added is the first one to be removed.
It's like a stack of plates, where we add a new plate to the top and remove the topmost plate when needed.

→ Basic Operation on Stack

- **push()**: To insert an element to the top of the stack
- **pop()**: To remove an element from the top of the stack
- **top()**: Returns the top element of the stack
- **isEmpty()**: Returns true if the stack is empty
- **size()**: Returns the size of stack

→ Implementation of Stack using Array

- A simple way of implementing the stack uses an array.
- A variable called top keep track of the index of the top element in the array



↳ Stack Class (Using Array)

```
public class Stack {
    private int top;
    private int stackArray[];

    // Constructor to initialize the stack
    public Stack (int size) {
        stackArray = new int[size];
        top = -1; // This mean that no element
                  exist in any cell
    }
```

```
// Check if the stack is empty
public boolean isEmpty (){
    return (top == -1);
}

// Check if the stack is Full (the stack is full
  when top is equal to the index of the last
  cell in the array)
public boolean isFull (){
    return (top == stackArray.length -1);
}

// Push operation to add an element to the top
public void push (int item){
    if (!isFull()) {
        top++;
        stackArray[top] = item;
    }
    else
        System.out.println ("The Stack is Full");
}

// Pop Operation to remove the element from the top
public int pop (){
    if (isEmpty()){
        System.out.println ("The stack is Empty");
        System.exit(1);
    }
    return stackArray[top--];
}
```

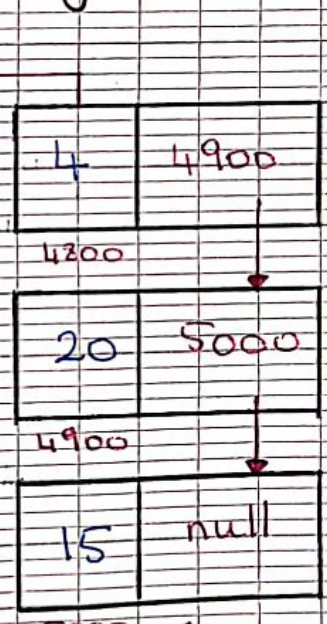
```
// Top operation to retrieve the top element
   without removing it
   public int top() {
        if (isEmpty()) {
             System.out.println("The stack is Empty");
             System.exist(1);
        }
        return stackArray[top];
   }
```

→ **Implementation of Stack using LinkedList**

- In a stack implemented using a linked list, we leverage the dynamic structure of a LinkedList to manage the elements.

- To implement a Stack using a LinkedList, all the linked list operations should be performed based on Stack operations LIFO

!! we will use the same classes NodeData and Node from the LinkedList



Initial stack having three element and top have address 4800

↳ Stack Class (Using LinKed List)

```
public class Stack {
  private Node top;
  .
  .
  // Constructor to initialize the Stack
  public Stack (){
      top = null;
  }
  .
  .
  .
  // Check if the stack is Empty
  public boolean isEmpty (){
    return (top == null);
```

↳ Push() Operation

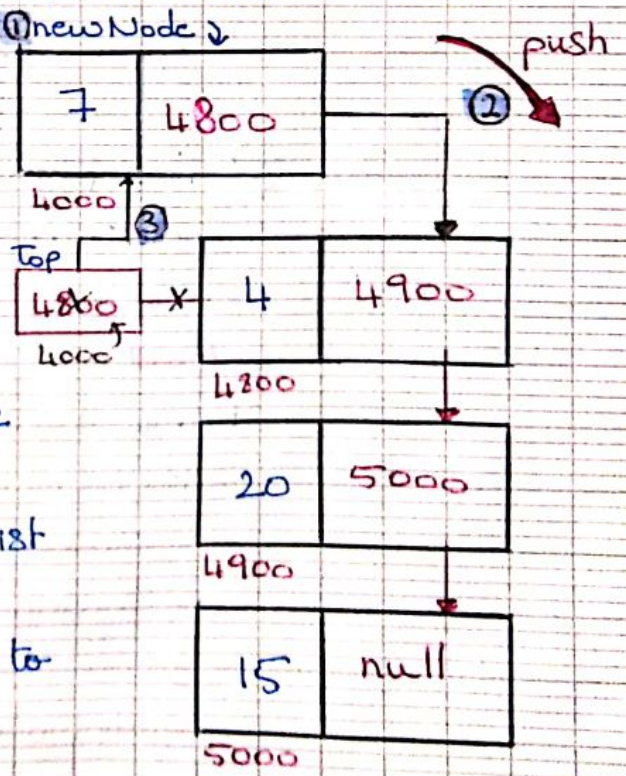
!! Same as addHead() in class LinKed List

. Approach:

① Create a newNode

② LinK the newNode to the top of the linked list by setting the 'next' reference of this newNode to point to the top of the linked List

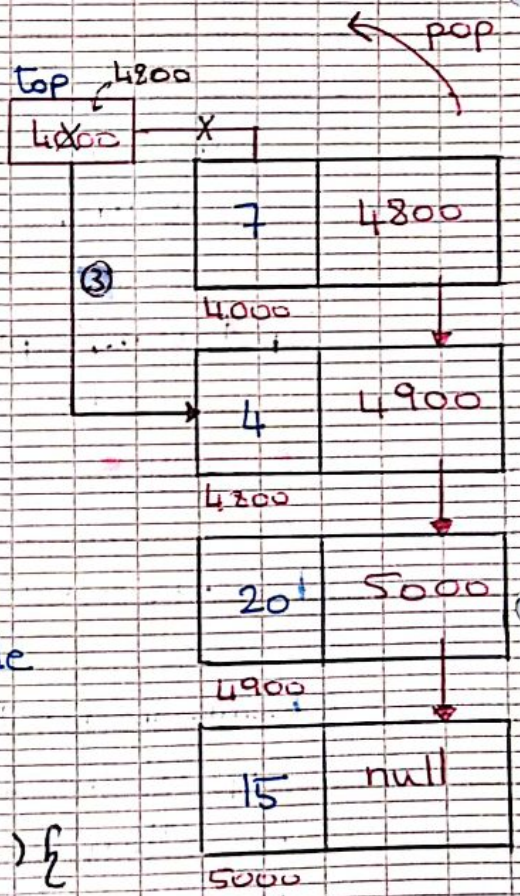
③ Update the 'top' reference to point to the newNode



. Code:

```
// Push an element at the top
public void push (NodeData item) {
    Node newNode = new Node (item);
    newNode.next = top;
    top = newNode;
}
```

↳ Pop() Operation



. Approach:

① Check whether the stack is Empty

② If not, the value of the top is returned

③ The 'top' reference will point to the next node

. Code:

```
Public NodeData pop () {
    if (isEmply ()) {
        System.out.println("The stack is Empty");
        System.exit(1);
    }
    NodeData nd = top.data; // store the top element
    top = top.next; // delete the top node
    return nd;
}
```

↳ Operation top()

This operation returns the top element without removing it

```
Public NodeData top (){
    if (isEmpty ()){
        System.out.println ("The Stack is Empty");
        System.exist (1);
    }
    return top.data;
```

↳ Display ()

```
// Display the content of the Stack
public void display () {
    Node current = top;
    while (current != null){
        System.out.println (current.data.value + " ");
        current = current.next;
    }
}
```

↳ CountNodes ()

```
// Count the number of Nodes in the stack
public int countNodes (){
    int count = 0;
    Node current = top;
    while (current != null){
        count ++;
        current = current.next;
    }
    return count;
}
```