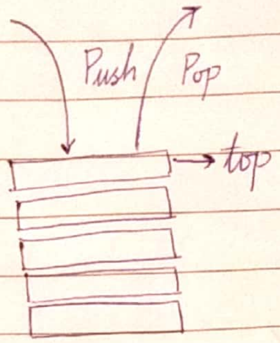
Day-1.

___/___/___

Stack Data Structure



last in first out.

(the order may be LIFO or FILO)

→ Stack is a linear data structure
→ Orders may be LIFO or FILO

Basic Operations on Stack – • Push
• Pop
• Peek or Top
• isEmpty
• isFull

} all take O(1) ⇓ we do not run any loop

1. Push: Adds an item in the stack. If Stack isfull → it is said to be Overflow condition.
2. Pop: Removes an item from stack. The items are popped in reverse order in which they are pushed. If Stack isempty → it is said to be Underflow condition.
3. Peek or Top: Returns top element of stack.
4. isEmpty: Returns true if stack is empty, else false.
5. isFull: Returns true if stack is full, else false.

Applications of Stack – Balancing of Symbols, Infix to Postfix/Prefix,

Redo/Undo features, forward/backward in Websites
Algorithms like – Tower of Hanoi, Tree traversals, Stock Span Problem, Histogram Problem.
Applications – Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver.
Graph Algorithms – Topological Sorting and Strongly Connected Components

Two ways to implement Stack -

1. Using Array
2. Using Linked List

1. Using Array.

```
class Stack{
    static final int MAX = 1000;
    int top;
    int a[] = new int[MAX]; //max size of ~~array~~ stack.

    boolean isEmpty(){
          return(top<0);
     }
      stack(){
       top=-1;
      }
       boolean push(int x){
         if(top>=(MAX-1)){
              System.out.println("Stack Overflow");
              return false;
           }
          else{
             a[++top] = x;
             System.out.println(x+"pushed into stack");
             return true;
           }
        }

      int pop(){
            if(top<0){
                System.out.println("Stack Underflow");
                return 0;
               }
```

pre-increment

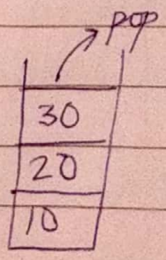
```
top = top + 1;
~~ster~~ a[top] = x;
```

```
else {
   int x = a[top--];
   return x;
  }
 }
 int peek() {
     if (top<0) {
         System.out.println("Stack Underflow");
         return 0;
     }
     else {
      int x = a[top];
      return x;
     }
   }
}
 class Main {
    public static void main(String args[]) {
            Stack s = new Stack();
            s.push(10);
            s.push(20);
            s.push(30);
            System.out.println(s.pop() + "Popped from stack");
     }
  }
```

post-decrement

```
x = a[top]
top = top-1
```

```
O/P:  10 pushed into stack
      20  "    "    "
      30  "    "    "
      30 popped from stack
```

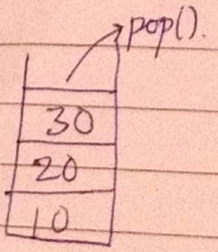


2. Using LinkedList.

```
public class Stack {
    StackNode root;
    static class StackNode {
        int data;
        StackNode next;   ~~next~~
        StackNode(int data) {
            this.data = data;
        }                *(next = null)
    }
    public boolean isEmpty() {
        if (root == null) {
            return true;
        }
        else
            return false;
    }
    public void push(int data) {
        StackNode newNode = new StackNode(data);
        if (root == null) {
            root = newNode;
        }
        else {
            StackNode temp = root;
            root = newNode;
            ~~root~~. newNode.next = temp;
        }
        System.out.println(data + " pushed to stack");
    }
    public int pop(){
        int popped = Integer.MIN_VALUE;
        if(root == null){
            System.out.println("Stack is Empty");
        }
        else{
            popped = root.data;
            root = root.next;
        }
        return popped;
    }
    public int peek(){
        if(root == null){
            System.out.println("Stack is empty");
            return Integer.MIN_VALUE;
        }
        else{
            return root.data;
        }
    }
    public static void main(String[]args){
        Stack s = new Stack();
            s.push(10);
            s.push(20);
            s.push(30);
        System.out.println(s.pop()+" popped from stack");
        System.out.println("Top element is "+s.peek());
    }
}
```

O/p:-

```
10 pushed to stack
20 "  "  "
30 "  "  "
30 popped from stack
~~Popped~~ Top element is 20
```



1. Assay Implementation:-

Pros - Easy to implement
- Memory saved as pointers not involved.

Cons - Not dynamic
- Doesn't grow/shrink depending on runtime.

2. Linked-List Implementation:-

Pros - Dynamic
- Can grow/shrink depending on runtime.

Cons - Extra Memory - due to involvement of pointers

⇒ Linked-List Basic Implementation Class. (for reference):-

```
class LinkedList {
    Node head;
    class Node {
        int data;
        Node next;
        Node (int d) {
            data = d;
            next = null;
        }
    }
}
```

— End of Challenge Day - 1. —