2022
AUGUST
1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31
M T W T F S S M T W T F S S

July
29th Wk • 197-168
16
Saturday

STACK

There are 3 major operations in STACK -

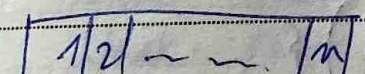
(i) Push O(1) [∵ O(1) is the time complexity].
(ii) Pop O(1)
(iii) Peek (O(1)) → to view the value of top element.

It is a LIFO (Last In First Out) Data Structure.

Implementation

Array

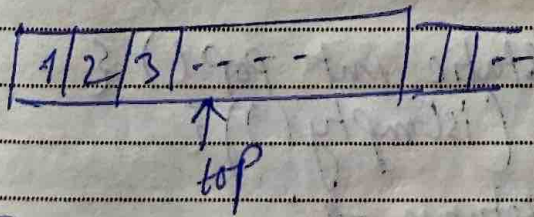
✓ It has fixed sized (n)



Stack is full when (arr el = = n). It's a hectic process, so we shall not often use array to implement stack.

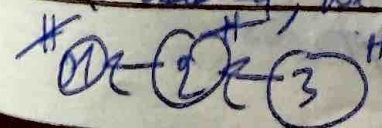
Arraylist

✓ It has dynamic size / variable size. So, elements can keep on getting added and the arraylist keeps on increasing it's size.



Linked list

It has also variable size. As soon as new elements are ~~element~~ added, we shall keep on changing the head.



17 July
Sunday
29th Wk • 198-167

2022 207

Implementing Stack using Linked List

```
Public class StackDS {
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            next = null;
        }}
    Static class Stack {
        Public Static Node head = null;
        Public Static void push(int data) {
            ~~Node~~ Node newNode = new Node(data);

            if (head == null) {
                head = newNode;
                return;
            }
            newNode.next = head;
            head = newNode;
        }
        Public static boolean isEmpty() {
            return (head == null);
        }
        public static int pop() {
            if (isEmpty()) {
                return -1;
            }
            Node top = head;
            head = head.next;
            return top.data;
        }
```

```
Public static int peek() {
    if (isEmpty())
    {
        return -1;
    }
    Node top = head;
    return top.data;
}
}

Public static void main (String args[])
{
    Stack stack = new Stack();
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);

    while (! stack.isEmpty())
    {
        System.out.println(stack.peek());
        stack.pop();
    }
}
}
```

Note :- pop() takes the uppermost element in the stack and pop it out of it.

peek() gets the value of a particular ~~element~~ index of the stack.

19 July
Tuesday
30th Wk • 200-165

2022

JULY

Implementing stack using Array List

```
import java.util.ArrayList;

Public class stackAL {
    Static class Stack {

ArrayList <Integer> list = new ArrayList<>();

Public void Push (int data) {
    list.add (data);
}

Public boolean isEmpty() {
    return (list.size == 0);
}

Public int Pop() {
    if (isEmpty())
    {
        return -1;
    }
int top = list.remove (list.size()-1);
return top;
}

Public int Peek() {
    if (isEmpty ()) {
        return -1;
    }
    return list.get (list.size()-1);
}
}
```

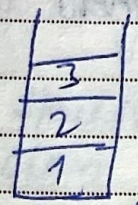
```
Public static void main (String args[])
{
    Stack stack = new Stack();

    Stack.Push(1);
    Stack.Push(2);
    Stack.Push(3);
    Stack.push(4);

while (! stack.isEmpty()) {

System.out.println(stack.peek());
    Stack.Pop();
}
}
}
```

** Implementing Stack Using Java Collection Framework **

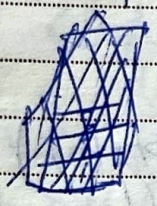
```
import java.util.*;

Public class StackJCF {
    Public static void main (String args[])
    {
Stack<Integer> stack = new Stack<>();
Stack.Push(1);
Stack.Push(2);
Stack.Push(3);
Stack.Push(4);

while (! stack.isEmpty()) {
    System.out.println(stack.peek());
    Stack.Pop();
}}
```

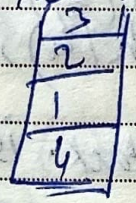
(Q1) Push at the bottom of the Stack

E.g, old stack



add element 4 to the stack.





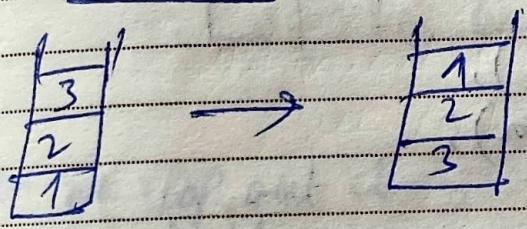
→ new stack

Algorithm → Remove all elements
Push the new element
Then push all the removed elements.

```
import java.util.*
Public class Stackproblem1 {
Public static void PushAtBottom (Stack<Integer> s, int data)
    {
        if(s.isEmpty()) {      //isEmpty() is same as before
            s.push(data);
            return;
        }
    int temp = s.Pop();
    PushAtBottom(s, data);
    }s.Push(temp);

    Public Static void main(String args[])
    {
    Stack<Integer> Stack = new Stack<>();
    Stack.Push(1); Stack.Push(2); Stack.Push(3);
    PushAtBottom(4);
```

```
while ( ! Stack.isEmpty() )
{
System.out.println( Stack.Pop() );
}
}
}
```

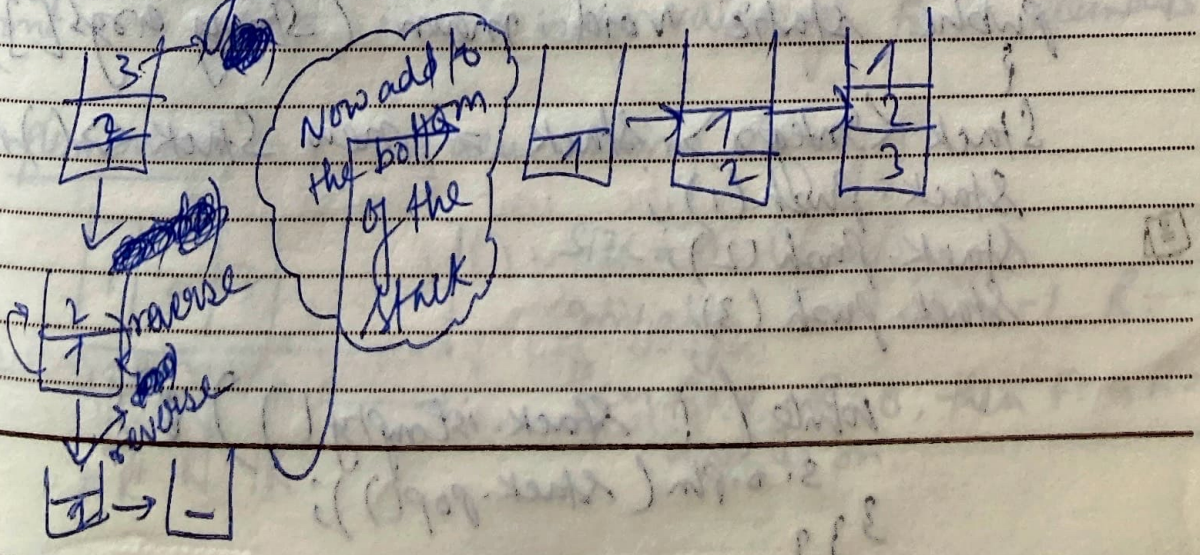
Q2) Reverse a Stack



Algorithm :- There are 2 approaches that can be followed.

Approach 1 :- Empty the Stack and Pop every elements one by one into a new Stack.

Approach 2 :- Reverse Sub-Stacks and then when we reach null Stack, add every to the bottom of the Stack.



23 July
Saturday
30th Wk • 204-161

2022

```
import java.util.*;

Public class StackProblem2 {

Public static void PushAtBottom (Stack<Integer> s, int data)
    {
        if (s.isEmpty())
        {
            s.push(data);
            return;
        }
        int temp = s.pop();
        PushAtBottom (s, data);
        s.push (temp);
    }
Public static void reverse (Stack<Integer> s)
    {
        if (s.isEmpty()) {
            return;
        }
        int top = s.pop();
        reverse(s);
        PushAtBottom (s, top);
    }
    public static void main (String args[])
    {
    Stack<Integer> stack = new Stack<>();
    stack.push(1);
    stack.push(2);
    stack.push(3);

    while (! stack.isEmpty()) {
        s.o.println (stack.pop());
    }}}
```