

July

29th Wk • 197-168

16  
Saturday

STACK

8 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.

## 12 Implementation

Aosay

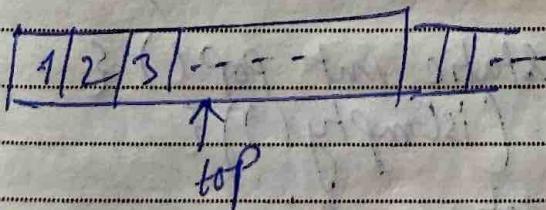
✓ It has fixed size (n)

121 ~ ~ 1n

Stack is full when ( $\text{arr el} = \pm n$ ). 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 its size.



## linked list

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

17

July

29th Wk. • 198-167

Sunday.

2022

207

MTWTFSS	SMTWTFSS
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	

## Implementing Stack using linked list

→ 8 Public class StackDS {

    private static class Node {

        9     int data;

        node next;

        Node(int data) {

            this.data = data;

            next = null;

    12     }

    Static class Stack {

        Public static Node head = null;

        1     Public static void push (int data) {

~~Node~~ Node newNode = new Node(data);

            2     if (head == null) {

                3         head = newNode;

                return;

                4         newNode.next = head;

                5         head = newNode;

        6     Public static boolean isEmpty () {

            return (head == null);

        7     Public static int pop () {

            8     if (isEmpty ()) {

                9         return -1;

                10      Node top = head,

                11      head = head.next;

                12      return top.data;

AUGUST  
 2022  
 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

Mo	Tu	We	Th	Fr	Sa	Su
15	16	17	18	19	20	21

July  
 30th Wk • 199-166

18  
 Monday

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

30th Wk • 200-165

Tuesday

SU	MO	TU	WE	TH	FR	SU	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	F	S	S	M	T	W	F	S	S	M	T	W	F	S	S	M	T	W	F	S	S	M	T	W	F	S	S	M	T	W	F	S	S								
2022	2023																																										

## \* \* Implementing stack using ArrayList

```

→8 import java.util.ArrayList;
9  public class Stack {
10    static class Stack {
11      ArrayList<Integer> list = new ArrayList<>();
12      public void Push (int data) {
13        list.add (data);
14      }
15      public boolean isEmpty() {
16        return (list.size == 0);
17      }
18      public int Pop() {
19        if (isEmpty())
20          return -1;
21        int top = list.remove (list.size() - 1);
22        return top;
23      }
24      public int peek() {
25        if (isEmpty())
26          return -1;
27        return list.get (list.size() - 1);
28      }
29    }
30  }

```

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

2022

July  
30th Wk • 201-164

20  
Wednesday

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> stack22 = new Stack<>();

    stack.push(1);

    stack.push(2);

    stack.push(3);

    stack.push(4);

    while ( ! stack.isEmpty() ).{

        System.out.println ( stack.peek() );

        stack.pop();

21

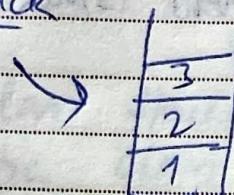
July

30th Wk • 202-163

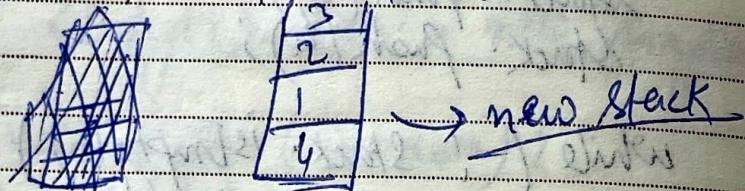
Thursday

(ii) Push at the bottom of the stack

8 E.g., old stack



10 add element 4 to the stack



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

2 → import java.util.\*;

3 Public class Stackproblem {

```
    public static void PushAtBottom (Stack<Integer> S, int data)
    {
```

if (~~s~~ s.isEmpty()) { // isEmpty() is same as before  
s.push('P' + i);

3. 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);
pushWithBottom(4);
```

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

July

30th Wk • 203-162

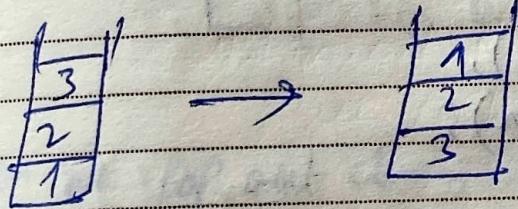
22

Friday

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

System.out.println(stack.pop())

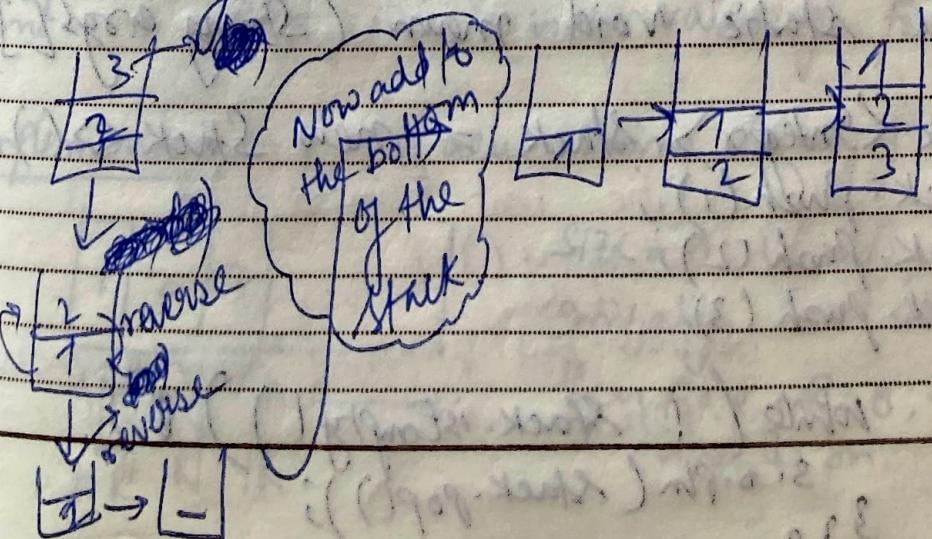
### (82) Reverse a Stack



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

30th Wk • 204-161

2022

213

Saturday

SUN	MON	TUE	WED	THU	FRI	SAT	SUN
BY	51	57	11	1	2	3	4
11	12	13	14	15	16	17	18
25	26	27	28	29	30	31	1
2	3	4	5	6	7	8	9
20	21	22	23	24	25	26	27

→ import java.util.\*;

```

8 Public class StackProblem2 {
9     Public static void PushAtBottom (Stack<Integer> s, int data) {
10    if (!s.isEmpty())
11        s.push(data);
12    return;
13    int temp = s.pop();
14    PushAtBottom (s, data);
15    s.push(temp);
}

```

2 Public static void reverse (Stack<Integer> s)

```

3 if (s.isEmpty()) {
4     return;
5     int top = s.pop();
6     reverse(s);
7     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()) {
    System.out.println(stack.pop());
}

```