

1. What is a Stack?

A **Stack** is a linear data structure that follows the **LIFO (Last In First Out)** principle. The last inserted element is removed first.

Stack in Java (Collection Framework)

```
Stack<Integer> st = new Stack<>();
```

Important Stack Methods

Method	Meaning
push (x)	Add an element
pop ()	Remove + return the top element
peek ()	Return top element without removing
isEmpty ()	Check if stack is empty
size ()	Returns count of elements
search (x)	Finds 1-based position from top

Sample Code – Basic Stack Operations

```
import java.util.Stack;

public class StackBasics {
    public static void main(String[] args) {

        Stack<Integer> st = new Stack<>();

        st.push(10);
        st.push(20);
        st.push(30);

        System.out.println("Stack = " + st);
        System.out.println("Top = " + st.peek());
        System.out.println("Popped = " + st.pop());
        System.out.println("Size = " + st.size());
        System.out.println("Is Empty = " + st.isEmpty());
    }
}
```

IMPORTANT STACK PROBLEMS

Problem 1: Push Numbers in Stack & Print All

Input: push 1, 2, 3

Output: 3 2 1

```

Stack<Integer> st = new Stack<>();
st.push(1);
st.push(2);
st.push(3);

while(!st.isEmpty())
    System.out.print(st.pop() + " ");

```

Problem 2: Reverse a String

```

String s = "hello";
Stack<Character> st = new Stack<>();

for(char c : s.toCharArray()) st.push(c);

String rev = "";
while(!st.isEmpty()) rev += st.pop();

System.out.println(rev);

```

Problem 3: Find Minimum Element in Stack

```

Stack<Integer> st = new Stack<>();
st.push(5);
st.push(1);
st.push(3);

int min = Integer.MAX_VALUE;

for(int x : st)
    min = Math.min(min, x);

System.out.println("Min = " + min);

```

Problem 4: Check if Stack is Sorted (top → bottom)

```

static boolean isSorted(Stack<Integer> st) {
    int prev = Integer.MAX_VALUE;

    for(int x : st) {
        if(x > prev) return false;
        prev = x;
    }
    return true;
}

```

Problem 5: Count Elements in Stack

```
System.out.println("Count = " + st.size());
```

Problem 6: Remove All Even Numbers

```

Stack<Integer> st = new Stack<>();
Stack<Integer> temp = new Stack<>();

for(int i : st)
    if(i % 2 != 0)
        temp.push(i);

```

```
st = temp;
```

Problem 7: Display Middle Element of Stack

```
int mid = st.size() / 2;
System.out.println("Middle = " + st.get(mid));
```

Problem 8: Sum of All Elements in Stack

```
int sum = 0;
for(int x : st) sum += x;

System.out.println(sum);
```

Problem 9: Duplicate the Entire Stack

Example: [1,2,3] → [1,2,3,1,2,3]

```
Stack<Integer> newStack = new Stack<>();

newStack.addAll(st);
newStack.addAll(st);
```

Problem 10: Print Stack from Bottom to Top (without removing)

```
for(int i = 0; i < st.size(); i++)
    System.out.print(st.get(i) + " ");
```

STACK IMPLEMENTATION IN JAVA

Stack Using Array

```
class ArrayStack {

    int top;
    int[] arr;

    ArrayStack(int size) {
        arr = new int[size];
        top = -1;
    }

    void push(int x) {
        if(top == arr.length - 1) {
            System.out.println("Overflow");
            return;
        }
        arr[++top] = x;
    }

    int pop() {
        if(top == -1) {
            System.out.println("Underflow");
            return -1;
        }
        return arr[top--];
    }
}
```

```

        int peek() {
            return (top == -1) ? -1 : arr[top];
        }

        boolean isEmpty() {
            return top == -1;
        }
    }
}

```

Stack Using Linked List

```

class Node {
    int data;
    Node next;
    Node(int d) { data = d; }
}

class LinkedStack {

    Node top;

    void push(int x) {
        Node n = new Node(x);
        n.next = top;
        top = n;
    }

    int pop() {
        if(top == null) {
            System.out.println("Underflow");
            return -1;
        }
        int val = top.data;
        top = top.next;
        return val;
    }

    int peek() {
        return (top == null) ? -1 : top.data;
    }

    boolean isEmpty() {
        return top == null;
    }
}

```