Day 20: Implement Stack

Abhinav Yadav

"Think like a stack: handle the last task first."
— Anonymous

---

1 Introduction

A **Stack** is a linear data structure that follows the **Last In First Out (LIFO)** principle. The most recently added element is the first to be removed. It supports the following operations:

- **Push:** Add an element to the top of the stack.
- **Pop:** Remove the top element of the stack.
- **Peek/Top:** View the top element without removing it.

2 Applications of Stack

- Expression evaluation and conversion (e.g., infix to postfix).
- Backtracking algorithms (e.g., navigating a maze).
- Function call management in recursion.

3 Code

```
class Stack {
    private static final int MAX = 100; // Maximum size of the
       stack
    private int top;
    private int[] data;

    // Constructor to initialize the stack
    public Stack() {
        top = -1;
        data = new int[MAX];
    }
```

```
11
12      // Push operation
13      public void push(int value) {
14          if (top == MAX - 1) {
15              System.out.println("Stack Overflow");
16              return;
17          }
18          data[++top] = value;
19
20          // Display the stack after the push
21          System.out.print("Stack after push: ");
22          for (int i = 0; i <= top; i++) {
23              System.out.print(data[i] + " ");
24          }
25          System.out.println();
26      }
27
28      // Pop operation
29      public int pop() {
30          if (top == -1) {
31              System.out.println("Stack Underflow");
32              return -1;
33          }
34          int poppedValue = data[top--];
35
36          // Display the stack after the pop
37          System.out.print("Stack after pop: ");
38          for (int i = 0; i <= top; i++) {
39              System.out.print(data[i] + " ");
40          }
41          System.out.println();
42
43          return poppedValue;
44      }
45
46      // Main function to test stack operations
47      public static void main(String[] args) {
48          Stack stack = new Stack(); // Initialize the stack
49
50          // Push elements onto the stack
51          stack.push(5);
52          stack.push(10);
53          stack.push(15);
54
55          // Pop elements from the stack
56          System.out.println("Popped: " + stack.pop()); // Should
                print 15
57          System.out.println("Popped: " + stack.pop()); // Should
                print 10
58          System.out.println("Popped: " + stack.pop()); // Should
                print 5
```

```
59
60          // Try popping from an empty stack
61          System.out.println("Popped: " + stack.pop()); // Should
               print "Stack Underflow"
62      }
63  }
```

4 Stack operations: Visual Representation and Output

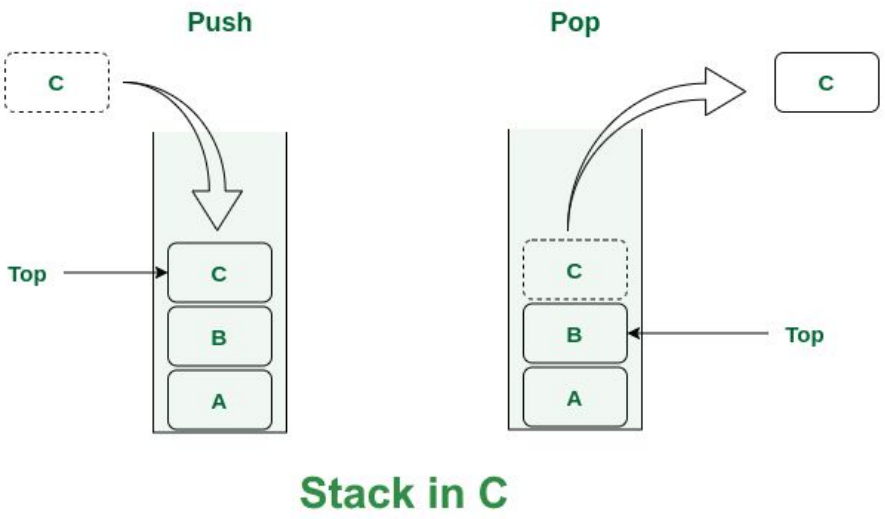

Figure 1: Stack Operations: Push and Pop

5 Conclusion

The stack data structure is an essential tool for implementing algorithms involving recursion, backtracking, and expression evaluation. Its simple LIFO approach is intuitive and effective for a wide range of use cases.

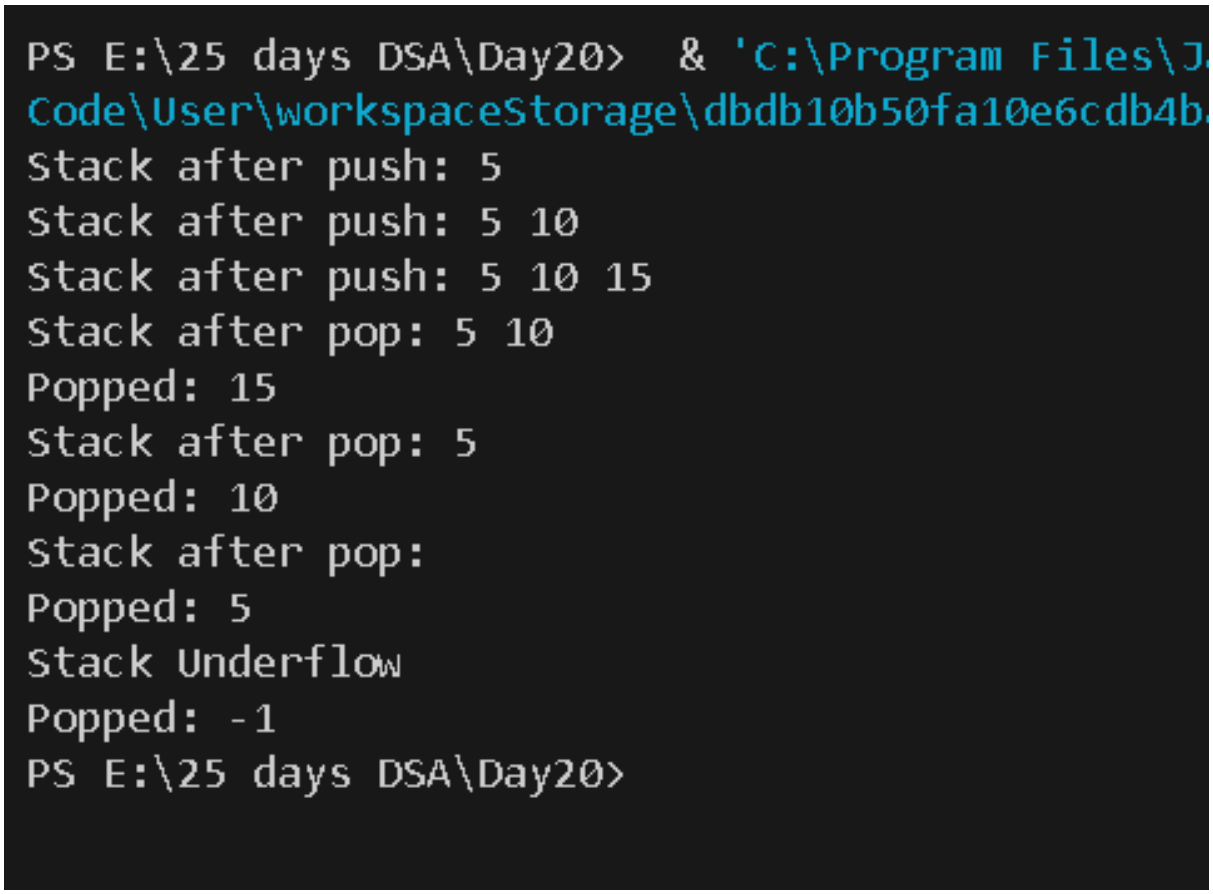

Figure 2: Program Output for Stack