Stack Implementation using Array

Generate DSA Notes for implement Stack using Array

DSA Notes: Implementing a Stack Using an Array

Introduction to Stack

- A stack is a linear data structure that follows the **Last In First Out (LIFO)** principle.
- Operations in a stack:
  - **Push:** Add an element to the top of the stack.
  - **Pop:** Remove the top element of the stack.
  - **Peek/Top:** Retrieve the top element without removing it.
  - **isEmpty:** Check if the stack is empty.
  - **isFull (optional):** Check if the stack has reached its maximum capacity (for fixed-size stacks).

Key Characteristics of Stack Using Array

1. **Static Memory Allocation:** Predefine the maximum size of the stack.
2. **Indexes for Operations:** Use an index (e.g., `top`) to track the top of the stack.
   - Initialize `top = -1` (empty stack).
   - Increment `top` for **push**.
   - Decrement `top` for **pop**.

Steps to Implement Stack Using Array

1. **Define a fixed-size array.**
2. **Maintain a `top` pointer** initialized to `-1`.

3. **Implement key operations:**
    - **Push Operation:**
        - Check if the stack is full (`top == size - 1`).
        - Increment `top` and add the element at `stack[top]`.
    - **Pop Operation:**
        - Check if the stack is empty (`top == -1`).
        - Remove the element at `stack[top]` and decrement `top`.
    - **Peek Operation:**
        - Return the element at `stack[top]` if the stack is not empty.
    - **isEmpty Operation:**
        - Return `true` if `top == -1`.

Implementation

Here's a Java-based example implementation:

java

```java
class Stack {
    private int[] stackArray; // Array to hold stack elements
    private int top;          // Index of the top element
    private int maxSize;      // Maximum size of the stack

    // Constructor
    public Stack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1; // Initialize stack as empty
    }

    // Push operation
    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack Overflow! Cannot push " + value);
```

```
            return;
        }
        stackArray[++top] = value; // Increment top and insert value
    }

    // Pop operation
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow! Cannot pop");
            return -1;
        }
        return stackArray[top--]; // Return top value and decrement top
    }

    // Peek operation
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }
        return stackArray[top];
    }

    // Check if stack is empty
    public boolean isEmpty() {
        return top == -1;
    }

    // Check if stack is full
    public boolean isFull() {
        return top == maxSize - 1;
    }
}
```

Example Usage

java

```
public class StackDemo {
    public static void main(String[] args) {
        Stack stack = new Stack(5);

        // Push elements
        stack.push(10);
        stack.push(20);
        stack.push(30);

        // Display top element
        System.out.println("Top element: " + stack.peek());

        // Pop elements
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Top element after pop: " + stack.peek());

        // Check if stack is empty
        System.out.println("Is stack empty? " + stack.isEmpty());
    }
}
```

Time Complexity

1. **Push Operation:** $O(1)$
2. **Pop Operation:** $O(1)$
3. **Peek Operation:** $O(1)$

Advantages of Stack Using Array

1. **Simple Implementation:** Easy to use for fixed-size stacks.
2. **Efficient Operations:** All operations have $O(1)$ time complexity.

Limitations of Stack Using Array

1. **Fixed Size:** You need to specify the maximum size during initialization.
2. **Space Inefficiency:** If the stack is not full, unused space remains allocated.

Key Learning Points

- Understand how to use an array and pointer (`top`) to simulate stack operations.
- Handle edge cases like stack overflow and underflow.
- Learn the trade-offs of using an array for stack implementation versus dynamic alternatives like linked lists.