Lecture 01: Stack Data Structure

Introduction

A stack is a fundamental data structure that follows a specific order for operations. It allows access to only one data item - the last item inserted. Once you remove this item, you can access the next-to-last item inserted.

Key Characteristics

- The stack data structure follows a Last-In-First-Out (LIFO) principle, meaning the last element added is the first one to be removed.
- All insertions and deletions occur at one end of the stack, known as the "Top."
- Operations in the middle or at other ends of the stack are restricted.
- Elements are removed in the reverse order of their insertion.

Basic Operations

1. Push Operation

- **Definition**: Adding an item to the stack
- **Process**: The new item is placed at the top of the stack
- **Key Point**: If the stack is full, a stack overflow condition occurs

2. Pop Operation

- **Definition**: Removing an item from the stack
- **Process**: The topmost item is removed from the stack
- **Key Point**: If the stack is empty, a stack underflow condition occurs

3. Peek Operation

- **Definition**: Viewing the top item without removing it
- **Process**: Returns the value of the top element
- **Key Point**: Only the top item can be viewed; all other items are invisible to the user

Real-World Applications

1. String Reverse
2. Web Browser History
3. Undo Operations in Text Editors
4. Recursive Function Calls
5. Supporting Data Structure for Various Algorithms

Technical Implementation

Basic Stack Class Structure

```
class StackX {
    private int maxSize;    // size of stack array
    private double[] stackArray;
    private int top;        // top of the stack

    public StackX(int s) {  // constructor
        maxSize = s;        // set array size
        stackArray = new double[maxSize];
        top = -1;          // no items yet
    }
}
```

Core Operations Implementation

Push Implementation

```
public void push(double j) {
    // check whether stack is full
    if (top == maxSize – 1)
        System.out.println("Stack is full");
    else
        stackArray[++top] = j;
}
```

Pop and Peek Implementation

```
public double pop() {
    if (top == -1)
        return -99;
    else
        return stackArray[top--];
}

public double peek() {
    if (top == -1)
        return -99;
    else
        return stackArray[top];
}
```

Practice Exercises

Exercise 1: Stack Frame Operations

Design a stack with a maximum size of 4 and implement basic stack operations.

Exercise 2: Implementation Challenge

Implement the following methods for a stack class:

- isEmpty(): Returns true if the stack has no elements
- isFull(): Returns true if the stack is at maximum capacity

Exercise 3: Stack Operations Practice

Create a program that:

1. Creates a stack with maximum size 10
2. Inserts the following items: 30, 80, 100, 25
3. Removes and displays all items

Example Solution for Exercise 2

```java
public class Stack {
    private int[] stack;
    private int top;
    private int capacity;

    public Stack(int capacity) {
        this.capacity = capacity;
        stack = new int[capacity];
        top = -1;
    }

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

    public boolean isFull() {
        return top == capacity - 1;
    }

    public boolean push(int item) {
        if (isFull()) {
            return false;
        }
        stack[++top] = item;
        return true;
    }

    public boolean pop() {
        if (isEmpty()) {
            return false;
        }
        top--;
        return true;
    }

    public int peek() {
        if (isEmpty()) {
            return Integer.MIN_VALUE;
        }
        return stack[top];
    }
}
```

Industrial Applications

- Stack operations are built into microprocessors
- Method calls in programming languages utilize stacks for:
    - Storing return addresses
    - Managing arguments
    - Handling method returns

This document serves as a comprehensive guide to understanding and implementing stack data structures in Java. It covers theoretical concepts, practical implementations, and real-world applications.