# Java Day13

## 🗂️ Stack (LIFO Structure)

- **Definition:** A stack is a collection of elements where insertion and deletion happen at one end only, called the *top*.
- **Principle: Last-In-First-Out (LIFO)** – the last element pushed is the first one popped.
- **Basic Operations:**
  - **push(x):** Insert element `x` at the top.
  - **pop():** Remove the top element.
  - **peek()/top():** View the top element without removing it.
  - **isEmpty():** Check if the stack is empty.
- **Implementation:**
  - Using **arrays** (fixed size).
  - Using **linked lists** (dynamic size).
- **Applications:**
  - Expression evaluation (Infix → Postfix conversion).
  - Function call management (recursion, nested calls).
  - Undo/Redo operations in editors.
  - Solving problems like *Towers of Hanoi*.

## 📦 Queue (FIFO Structure)

- **Definition:** A queue is a collection of elements where insertion happens at the *rear* and deletion happens at the *front*.
- **Principle: First-In-First-Out (FIFO)** – the first element enqueued is the first one dequeued.
- **Basic Operations:**
  - **enqueue(x):** Insert element `x` at the rear.
  - **dequeue():** Remove element from the front.

    - **peek()/front():** View the front element without removing it.
    - **isEmpty():** Check if the queue is empty.
- **Types of Queues:**
    - **Simple Queue:** Standard FIFO.
    - **Circular Queue:** Rear connects back to front to utilize space efficiently.
    - **Double-Ended Queue (Deque):** Insertion and deletion allowed at both ends.
    - **Priority Queue:** Elements dequeued based on priority rather than order.
- **Applications:**
    - Scheduling tasks (CPU scheduling, printer queue).
    - Managing requests in operating systems.
    - Breadth-First Search (BFS) in graphs.
    - Handling asynchronous data (network packets, messaging systems).

## 🔑 Key Differences Between Stack & Queue

| Feature | Stack (LIFO) | Queue (FIFO) |
|---|---|---|
| Access Order | Last in, first out | First in, first out |
| Insertion Point | Top only | Rear only |
| Deletion Point | Top only | Front only |
| Common Use Cases | Undo, recursion, expression evaluation | Scheduling, BFS, resource management |

## Implementation of Stack Using LinkedList

```
package DSA.Day13;

import java.util.NoSuchElementException;

/*
Linear DS
LIFO
Implementation: 1. Array 2. LinkedList 3. Queue
```

```
 */
class ListNode{
    int data;
    ListNode next;
    public ListNode(int data){
        this.data=data;
        this.next=null;
    }
}
public class StackDemo {
    private ListNode top;
    private int length;
    public StackDemo(){
        this.top=null;
        this.length=0;
    }
    public boolean isEmpty(){
        return length==0;
    }
    public int size(){
        return length;
    }
    public void push(int value){
        ListNode newNode=new ListNode(value);
        if(top==null){
            top=newNode;
            length++;
            return;
        }
        newNode.next=top;
        top=newNode;
        length++;
    }
    public int pop(){
        if(isEmpty()){
            throw new NoSuchElementException("Stack is Empty");
        }
        int result=top.data;
        top=top.next;
        length--;
        return result;
    }
    public int peek(){
        if(isEmpty()){
            throw new NoSuchElementException("Stack is Empty");
        }
        return top.data;
    }

    public static void main(String[] args) {
        StackDemo stack=new StackDemo();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        System.out.println(stack.size());
        System.out.println(stack.pop());
        System.out.println(stack.peek());
    }
}
```

## Implementation of Stack Using Queue

```

package DSA.Day13;

import java.util.NoSuchElementException;
import java.util.Queue;

/*
Linear DS
FIFO
Insertion end⇒ rear end
Deletion end⇒ front end
 */
```

```
public class QueueDemo {
    private ListNode front;
    private ListNode rear;
    private int length;
    public QueueDemo(){
        front=null;
        rear=null;
        length=0;
    }
    public boolean isEmpty(){
        return length==0;
    }
    public int size(){
        return length;
    }
    public void enqueue(int value){
        ListNode newNode=new ListNode(value);
        if(front==null){
            front=newNode;
        }else{
            rear.next=newNode;
        }
        rear=newNode;
        length++;
    }
    public int dequeue(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        int result=front.data;
        front=front.next;
        if(front==null){
            rear=null;
        }
        length--;
        return result;
    }
    public int first(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        return front.data;
    }
    public int last(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        return rear.data;
    }
    public static void main(String[] args) {
        QueueDemo queue=new QueueDemo();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        System.out.println(queue.dequeue());
        System.out.println(queue.dequeue());
    }
}
```

## Implementation of Queue Using LinkedList

```
package DSA.Day13;

import java.util.NoSuchElementException;
import java.util.Queue;

/*
Linear DS
FIFO
Insertion end⇒ rear end
Deletion end⇒ front end
 */
public class QueueDemo {
    private ListNode front;
    private ListNode rear;
    private int length;
    public QueueDemo(){
        front=null;
        rear=null;
        length=0;
    }
    public boolean isEmpty(){
        return length==0;
    }
    public int size(){
        return length;
    }
    public void enqueue(int value){
        ListNode newNode=new ListNode(value);
        if(front==null){
            front=newNode;
        }else{
            rear.next=newNode;
        }
        rear=newNode;
        length++;
    }
    public int dequeue(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        int result=front.data;
        front=front.next;
        if(front==null){
            rear=null;
        }
        length--;
        return result;
    }
    public int first(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        return front.data;
    }
    public int last(){
        if(isEmpty()){
            throw new NoSuchElementException("Queue is Empty");
        }
        return rear.data;
    }
    public static void main(String[] args) {
        QueueDemo queue=new QueueDemo();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        System.out.println(queue.dequeue());
        System.out.println(queue.dequeue());
    }
}
```

## Implementation of Queue Using Stacks

```
package DSA.Day13.Queue;

import java.util.Stack;

public class QueueImplementationUsingStacks {
    Stack<Integer> stack1;
    Stack<Integer> stack2;
    public QueueImplementationUsingStacks(){
        stack1=new Stack<>();
        stack2=new Stack<>();
    }
    public void enqueue(int value){
        stack1.push(value);
    }
    public int dequeue(){
        if(stack2.isEmpty()){
            while(!stack1.isEmpty()){
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }
    public int top(){
        if(stack2.isEmpty()){
            while(!stack1.isEmpty()){
                stack2.push(stack1.pop());
            }
        }
        return stack2.peek();
    }

    public static void main(String[] args) {
        QueueImplementationUsingStacks queue=new QueueImplementationUsingStacks();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        System.out.println(queue.dequeue());
    }

}
```