**Stacks**

***Stack Using Array***

public class StackUsingArray {

private int[] data;

private int topIndex;

public StackUsingArray() {

data = new int[10];

topIndex = -1;

}

// 1st method

public void push(int element) {

if (topIndex == data.length - 1) {

doubleCapacity();

}

data[++topIndex] = element;

}

// 2nd method

public int top() {

return data[topIndex];

}

// 3rd method

public int size() {

return topIndex + 1;

}

// 4th method

public boolean isEmpty() {

return topIndex == -1;

}

// 5th method

public int pop() {

int temp = data[topIndex];

data[topIndex] = 0;

topIndex--;

return temp;

}

// internal helper method

private void doubleCapacity() {

int[] temp = data;

data = new int[2 \* temp.length];

System.arraycopy(temp, 0, data, 0, temp.length);

System.out.println("Capacity of array: " + data.length);

}

}

***Stack Using LinkedList***

// Node Class

public class Node<T> {

public Node<T> next;

public T data;

public Node(T data) {

this.data = data;

}

}

// Linked List class

public class StackUsingLL<T> {

private Node<T> head;

private int size;

public StackUsingLL() {

head = null;

size = 0;

}

// 1st method

public void push(T elem) {

Node<T> newNode = new Node<T>(elem);

newNode.next = head;

head = newNode;

size++;

}

// 2nd method

public T top() {

return head.data;

}

// 3rd method

public T pop() {

T temp = head.data;

head = head.next;

size--;

return temp;

}

// 4th method

public int size() {

return size;

}

// 5th method

public boolean isEmpty() {

return size == 0;

}

}

// reverse a stack

public static void reverseStack(Stack<Integer> input, Stack<Integer> extra) {

// base condition

if (input.size() <= 1) return;

int lastElement = input.pop();

reverseStack(input, extra);

while (!input.isEmpty()) {

extra.push(input.pop());

}

input.push(lastElement);

while (!extra.isEmpty()) {

input.push(extra.pop());

}

}