stack using Array

```
/* Java program to implement stack
operations using array*/
class Stack
{
  static int MAX = 100;
  int top;
  int a[] = new int[MAX];     // Maximum size of Stack


  boolean isEmpty ()
  {
    return (top < 0);
  }
  Stack ()
  {
    top = -1;
  }

  boolean push (int x)
  {
    if (top >= (MAX - 1))
      {
        System.out.println ("Overflow condition reached");
        return false;
      }
    else
      {
        a[++top] = x;
        System.out.println (x + " pushed into stack");
        return true;
      }
  }


  int pop ()
  {
    if (top < 0)
      {
        System.out.println ("Underflow condition reached");
        return 0;
      }
    else
      {
        int x = a[top--];
        return x;
      }
  }


  int peek ()
  {
    if (top < 0)
      {
        System.out.println ("Underflow condition");
        return 0;
      }
```

```
    else
      {
        int x = a[top];
        return x;
      }
  }
}

class Main
{
  public static void main (String args[])
  {
    Stack stk = new Stack ();
      stk.push (20);
      stk.push (40);
      stk.push (60);
      System.out.println ("element poped out : " + stk.pop ());
  }
}
```

stack using ArrayList

```
// Java Program to Implement Stack in Java Using Array and
// Generics

// Importing input output classes
import java.io.*;
// Importing all utility classes
import java.util.*;

// user defined class for generic stack
class stack<T> {

	// Empty array list
	ArrayList<T> A;

	// Default value of top variable when stack is empty
	int top = -1;

	// Variable to store size of array
	int size;

	// Constructor of this class
	// To initialize stack
	stack(int size)
	{
		// Storing the value of size into global variable
		this.size = size;

		// Creating array of Size = size
		this.A = new ArrayList<T>(size);
	}

	// Method 1
	// To push generic element into stack
	void push(T X)
	{
		// Checking if array is full
		if (top + 1 == size) {

			// Display message when array is full
			System.out.println("Stack Overflow");
		}
		else {

			// Increment top to go to next position
			top = top + 1;

			// Over-writing existing element
			if (A.size() > top)
				A.set(top, X);

			else

				// Creating new element
				A.add(X);
		}
	}
```

```
// Method 2
// To return topmost element of stack
T top()
{
	// If stack is empty
	if (top == -1) {

		// Display message when there are no elements in
		// the stack
		System.out.println("Stack Underflow");

		return null;
	}

	// else elements are present so
	// return the topmost element
	else
		return A.get(top);
}

// Method 3
// To delete last element of stack
void pop()
{
	// If stack is empty
	if (top == -1) {

		// Display message when there are no elements in
		// the stack
		System.out.println("Stack Underflow");
	}

	else

		// Delete the last element
		// by decrementing the top
		top--;
}

// Method 4
// To check if stack is empty or not
boolean empty() { return top == -1; }

// Method 5
// To print the stack
// @Override
public String toString()
{

	String Ans = "";

	for (int i = 0; i < top; i++) {
		Ans += String.valueOf(A.get(i)) + "->";
	}

	Ans += String.valueOf(A.get(top));
```

```
            return Ans;
        }
}
// Main Class
public class GFG {

        // main driver method
        public static void main(String[] args)
        {

            // Integer Stack

            // Creating an object of Stack class
            // Declaring objects of Integer type
            stack<Integer> s1 = new stack<>(3);

            // Pushing elements to integer stack - s1

            // Element 1 - 10
            s1.push(10);
            // Element 2 - 20
            s1.push(20);
            // Element 3 - 30
            s1.push(30);

            // Print the stack elements after pushing the
            // elements
            System.out.println(
                "s1 after pushing 10, 20 and 30 :\n" + s1);

            // Now, pop from stack s1
            s1.pop();

            // Print the stack elements after popping few
            // element/s
            System.out.println("s1 after pop :\n" + s1);

            // String Stack

            // Creating an object of Stack class
            // Declaring objects of Integer type
            stack<String> s2 = new stack<>(3);

            // Pushing elements to string stack - s2

            // Element 1 - hello
            s2.push("hello");
            // Element 2 - world
            s2.push("world");
            // Element 3 - java
            s2.push("java");

            // Print string stack after pushing above string
            // elements
            System.out.println(
                "\ns2 after pushing 3 elements :\n" + s2);
```

```
        System.out.println(
                "s2 after pushing 4th element :");

        // Pushing another element to above stack

        // Element 4 - GFG
        s2.push("GFG");

        // Float stack

        // Creating an object of Stack class
        // Declaring objects of Integer type
        stack<Float> s3 = new stack<>(2);

        // Pushing elements to float stack - s3

        // Element 1 - 100.0
        s3.push(100.0f);
        // Element 2 - 200.0
        s3.push(200.0f);

        // Print string stack after pushing above float
        // elements
        System.out.println(
                "\ns3 after pushing 2 elements :\n" + s3);

        // Print and display top element of stack s3
        System.out.println("top element of s3:\n"
                                    + s3.top());
    }
}
```

stack using linked list

```
import java.lang.*;
public class Main {
    public static void main(String[] args) {
        Main Obj = new Main();
        Obj.insert(10);
        Obj.insert(20);
        Obj.insert(30);
        Obj.insert(40);
        Obj.insert(50);
        Obj.insert(60);

        System.out.println("Original List");
        Obj.print();

    }
    class Node{
        int element;
        Node next;
        Node prev;

        public Node(int element) {
            this.element = element;
        }
    }

    public Node head = null;
    public Node tail = null;
    int size=0;

    public void insert(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        newNode.prev = null;

        if (head != null)
            head.prev = newNode;

        head = newNode;
    }

    public void print() {

        Node node = head;
        Node end = null;

        while (node != null) {
            System.out.print(node.element + " ");
            end = node;
            node = node.next;
        }
        System.out.println();
    }
}
```

```
// Java program to Implement a stack
// using singly linked list
// import package
import static java.lang.System.exit;

// Driver code
class GFG {
	public static void main(String[] args)
	{
		// create Object of Implementing class
		StackUsingLinkedlist obj
			= new StackUsingLinkedlist();
		// insert Stack value
		obj.push(11);
		obj.push(22);
		obj.push(33);
		obj.push(44);

		// print Stack elements
		obj.display();

		// print Top element of Stack
		System.out.printf("\nTop element is %d\n",
						obj.peek());

		// Delete top element of Stack
		obj.pop();
		obj.pop();

		// print Stack elements
		obj.display();

		// print Top element of Stack
		System.out.printf("\nTop element is %d\n",
						obj.peek());
	}
}

// Create Stack Using Linked list
class StackUsingLinkedlist {

	// A linked list node
	private class Node {

		int data; // integer data
		Node link; // reference variable Node type
	}
	// create global top reference variable global
	Node top;
	// Constructor
	StackUsingLinkedlist() { this.top = null; }

	// Utility function to add an element x in the stack
	public void push(int x) // insert at the beginning
	{
		// create new node temp and allocate memory
```

```
	Node temp = new Node();

	// check if stack (heap) is full. Then inserting an
	// element would lead to stack overflow
	if (temp == null) {
		System.out.print("\nHeap Overflow");
		return;
	}

	// initialize data into temp data field
	temp.data = x;

	// put top reference into temp link
	temp.link = top;

	// update top reference
	top = temp;
}

// Utility function to check if the stack is empty or
// not
public boolean isEmpty() { return top == null; }

// Utility function to return top element in a stack
public int peek()
{
	// check for empty stack
	if (!isEmpty()) {
		return top.data;
	}
	else {
		System.out.println("Stack is empty");
		return -1;
	}
}

// Utility function to pop top element from the stack
public void pop() // remove at the beginning
{
	// check for stack underflow
	if (top == null) {
		System.out.print("\nStack Underflow");
		return;
	}

	// update the top pointer to point to the next node
	top = (top).link;
}

public void display()
{
	// check for stack underflow
	if (top == null) {
		System.out.printf("\nStack Underflow");
		exit(1);
	}
	else {
```

```
                Node temp = top;
                while (temp != null) {

                        // print node data
                        System.out.print(temp.data);

                        // assign temp link to temp
                        temp = temp.link;
                        if(temp != null)
                                System.out.print(" -> ");
                }
        }
    }
}
```

infix to postfix

```
import java.util.Stack;

public class InfixToPostfix {

	// Function to return precedence of operators
	static int prec(char c) {
		if (c == '^')
			return 3;
		else if (c == '/' || c == '*')
			return 2;
		else if (c == '+' || c == '-')
			return 1;
		else
			return -1;
	}

	// Function to return associativity of operators
	static char associativity(char c) {
		if (c == '^')
			return 'R';
		return 'L'; // Default to left-associative
	}

	// The main function to convert infix expression to postfix expression
	static void infixToPostfix(String s) {
		StringBuilder result = new StringBuilder();
		Stack<Character> stack = new Stack<>();

		for (int i = 0; i < s.length(); i++) {
			char c = s.charAt(i);

			// If the scanned character is an operand, add it to the output string.
			if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
				result.append(c);
			}
			// If the scanned character is an '(', push it to the stack.
			else if (c == '(') {
				stack.push(c);
			}
			// If the scanned character is an ')', pop and add to the output string from the stack
			// until an '(' is encountered.
			else if (c == ')') {
				while (!stack.isEmpty() && stack.peek() != '(') {
					result.append(stack.pop());
				}
				stack.pop(); // Pop '('
			}
			// If an operator is scanned
			else {
				while (!stack.isEmpty() && (prec(s.charAt(i)) < prec(stack.peek()) ||
                                                    prec(s.charAt(i)) == prec(stack.peek()) &&
                                                        associativity(s.charAt(i)) == 'L')) {
					result.append(stack.pop());
				}
				stack.push(c);
			}
```

```
            }

            // Pop all the remaining elements from the stack
            while (!stack.isEmpty()) {
                  result.append(stack.pop());
            }

            System.out.println(result);
      }

      // Driver code
      public static void main(String[] args) {
            String exp = "a+b*(c^d-e)^(f+g*h)-i";

            // Function call
            infixToPostfix(exp);
      }
}
```

infix to prefix

```
import java.util.*;
public class Main
{
    public static int precedence(char op) {
        switch (op) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
            case '%':
                return 2;
            case '^':
                return 3;
        }
        return -1;
    }

    public static String infixToPrefix(String infix) {
        String prefix = "";
        Stack< Character> operators = new Stack< >();

        for (int i = infix.length() - 1; i >= 0; --i) {
            char ch = infix.charAt(i);

            if (precedence(ch) > 0) {
                while (operators.isEmpty() == false &&  precedence(operators.peek()) > precedence(ch)) {
                    prefix += operators.pop();
                }
                operators.push(ch);
            } else if (ch == '(') {

                char x = operators.pop();
                while (x != ')') {
                    prefix += x;
                    x = operators.pop();
                }

            } else if (ch == ')') {
                operators.push(ch);
            } else {
                prefix += ch;
            }
        }

        while (!operators.isEmpty()) {
            prefix += operators.pop();
        }

        String reversedPrefix = "";
        for (int i = prefix.length() - 1; i >= 0; i--) {
            reversedPrefix += prefix.charAt(i);
        }
        return reversedPrefix;
    }
```

```
    public static void main (String[]args)
    {
        String exp = "A+B*(C^D-E)";
        System.out.println ("Infix Expression: " + exp);
        System.out.println ("Prefix Expression:" + infixToPrefix (exp));
    }
}
```