FA21-BCS-035  
M Barum Khan  
  
  
  
DSA LAB ASSIGNMENT 1

# Question1:

import java.util.Stack;

class QueueWithStack<T> {

    Stack<T> stack1 = new Stack();

    Stack<T> stack2 = new Stack();

    QueueWithStack() {

        stack1 = new Stack();

        stack2 = new Stack();

    }

    public void enqueue(T data) {

        stack1.push(data);

    }

    public void dequeue() {

        if (isEmpty()) {

            System.out.println("Queue is empty");

            return;

        }

        if (stack2.empty()) {

            while (!stack1.empty())

                stack2.push(stack1.pop());

        }

        System.out.println("Dequeued: " + stack2.pop());

    }

    public boolean isEmpty() {

        if (stack1.empty() && stack2.empty()) {

            return true;

        } else

            return false;

    }

}

public class Question1 {

    public static void main(String[] args) {

        QueueWithStack queue = new QueueWithStack<Integer>();

        queue.enqueue(1);

        queue.enqueue(5);

        queue.dequeue();

        queue.enqueue(1);

        queue.enqueue(5);

        queue.dequeue();

        queue.dequeue();

        queue.dequeue();

        queue.dequeue();

    }

}

# Question2:

import java.util.Scanner;

import java.util.Stack;

class ReversePolishNotation {

    Stack<Integer> stack = new Stack<Integer>();

    Scanner input = new Scanner(System.in);

    public int calculate() {

        System.out.println("Enter numbers or operators or ''quit'' (in lowercase) to exit");

        String choice = null; *// for expression*

        do {

            choice = input.next();

*// choice = choice.toLowerCase();*

            switch (choice) {

                case "+":

                    int a = stack.pop();

                    int b = stack.pop();

                    stack.push(b + a);

                    break;

                case "-":

                    a = stack.pop();

                    b = stack.pop();

                    stack.push(b - a);

                    break;

                case "\*":

                    a = stack.pop();

                    b = stack.pop();

                    stack.push(b \* a);

                    break;

                case "/":

                    a = stack.pop();

                    b = stack.pop();

                    a = b / a;

                    stack.push(a);

                    break;

                default:

                    stack.push(Integer.parseInt(choice));

                    break;

            }

            for (int i = 0; i < stack.size(); i++) {

                System.out.print(stack.get(i) + " ");

            }

        } while (choice != "quit");

        return stack.pop();

    }

}

public class Question2 {

    public static void main(String[] args) {

        ReversePolishNotation rpn = new ReversePolishNotation();

        rpn.calculate();

    }

}

# Question 3:

import java.util.Stack;

class InfixToPostfixConverter {

    String expression;

    private static boolean isOperator(char ch) {

        return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

    }

    private static int getPrecedence(char operator) {

        switch (operator) {

            case '+':

            case '-':

                return 1;

            case '\*':

            case '/':

                return 2;

            case '^':

                return 3;

        }

        return -1;

    }

    public static String convertToPostfix(String infix) {

        StringBuilder postfix = new StringBuilder();

        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < infix.length(); i++) {

            char ch = infix.charAt(i);

            if (Character.isLetterOrDigit(ch)) {

                postfix.append(ch);

            } else if (ch == '(') {

                stack.push(ch);

            } else if (ch == ')') {

                while (!stack.isEmpty() && stack.peek() != '(') {

                    postfix.append(stack.pop());

                }

                if (!stack.isEmpty() && stack.peek() != '(') {

                    return "Invalid infix expression";

                } else {

                    stack.pop();

                }

            } else if (isOperator(ch)) {

                while (!stack.isEmpty() && getPrecedence(ch) <= getPrecedence(stack.peek())) {

                    postfix.append(stack.pop());

                }

                stack.push(ch);

            }

        }

        while (!stack.isEmpty()) {

            if (stack.peek() == '(') {

                return "Invalid infix expression";

            }

            postfix.append(stack.pop());

        }

        return postfix.toString();

    }

}

public class Question3 {

    public static void main(String[] args) {

        InfixToPostfixConverter converter = new InfixToPostfixConverter();

        converter.expression = "a+b\*(c^d-e)";

        String infixExpression = converter.expression;

        String postfixExpression = InfixToPostfixConverter.convertToPostfix(converter.expression);

        System.out.println("Infix Expression: " + infixExpression);

        System.out.println("Postfix Expression: " + postfixExpression);

    }

}