الاسم : زهرة خالد

تقنية معلومات عربي

Topics

1. Create Stack Interface

```
public interface Stack<E> {
    boolean isEmpty();
    int size();
    E top();
    void push(E element);
    E pop();
}
```

2. Create Stack Using Array

```
public class ArrayStack<E> implements Stack<E> {
    private static final int DEFAULT_CAPACITY = 10;
    private E[] stackArray;
    private int top;

    public ArrayStack() {
        this(DEFAULT_CAPACITY);
    }

    public ArrayStack(int capacity) {
        if (capacity <= 0) {
            throw new IllegalArgumentException("Capacity must be
positive");
        }
        stackArray = (E[]) new Object[capacity];
        top = -1;
    }

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

    public int size() {
        return top + 1;
    }

    public E top() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return stackArray[top];
    }

    public void push(E element) {
        if (size() == stackArray.length) {
            resize(2 * stackArray.length);
        }
        stackArray[++top] = element;
    }

    public E pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        E element = stackArray[top];
        stackArray[top--] = null;
        if (size() > 0 && size() == stackArray.length / 4) {
            resize(stackArray.length / 2);
        }
        return element;
    }

    private void resize(int capacity) {
        E[] newArray = (E[]) new Object[capacity];
        for (int i = 0; i <= top; i++) {
            newArray[i] = stackArray[i];
        }
        stackArray = newArray;
    }
}
```

3. Create Stack Using Linked Lists

```
public class LinkedStack<E> implements Stack<E> {
    private Node<E> top;
    private int size;

    private static class Node<E> {
        private E element;
        private Node<E> next;

        public Node(E element, Node<E> next) {
            this.element = element;
            this.next = next;
        }
    }

    public LinkedStack() {
        top = null;
        size = 0;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int size() {
        return size;
    }

    public E top() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return top.element;
    }

    public void push(E element) {
        Node<E> newNode = new Node<>(element, top);
        top = newNode;
        size++;
    }

    public E pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        E element = top.element;
        top = top.next;
        size--;
        return element;
    }
}
```

4. Implement Basic Methods of Stack
    - isEmpty()
    - size()
    - top()
    - push(E e)
    - pop()

```
public class Main {
    public static void main(String[] args) {
        Stack<Integer> arrayStack = new ArrayStack<>();
        arrayStack.push(1);
        arrayStack.push(2);
        arrayStack.push(3);

        System.out.println(arrayStack.size());
        System.out.println(arrayStack.top());


        arrayStack.pop();
        System.out.println(arrayStack.size());
        System.out.println(arrayStack.top());

        Stack<String> linkedStack = new LinkedStack<>();
        linkedStack.push("Hello");
        linkedStack.push("World");

        System.out.println(linkedStack.size());
        System.out.println(linkedStack.top());

        linkedStack.pop();
        System.out.println(linkedStack.size());
        System.out.println(linkedStack.top());
    }
}
```

Homework

1. Implement a method with signature transfer(S, T) that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T.

```
public static <E> void transfer(Stack<E> S, Stack<E> T) {
    while (!S.isEmpty()) {
        T.push(S.pop());
    }
}
```

2. Give a recursive method for removing all the elements from a stack.

```
public static <E> void removeAll(Stack<E> stack) {
    if (!stack.isEmpty()) {
        stack.pop();
        removeAll(stack);
    }
}
```

الاسم : زهرة خالد

تقنية معلومات عربي

3. Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if "(exp1)op(exp2)" is a normal fully parenthesized expression whose operation is op, the postfix version of this is "pexp1 pexp2 op", where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of "((5 + 2) ∗ (8 − 3))/4" is "5 2 + 8 3 − ∗ 4 /". Describe a nonrecursive way of evaluating an expression in postfix notation.

```
import java.util.Stack;

public class PostfixEvaluator {
    public static int evaluatePostfix(String expression) {
        Stack<Integer> stack = new Stack<>();

        String[] tokens = expression.split(" ");

        for (String token : tokens) {
            if (isOperator(token)) {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                int result = performOperation(token, operand1, operand2);
                stack.push(result);
            } else {
                int operand = Integer.parseInt(token);
                stack.push(operand);
            }
        }

        return stack.pop();
    }

    private static boolean isOperator(String token) {
        return token.equals("+") || token.equals("-") || token.equals("*") ||
token.equals("/");
    }

    private static int performOperation(String operator, int operand1, int
operand2) {
        switch (operator) {
            case "+":
                return operand1 + operand2;
            case "-":
                return operand1 - operand2;
            case "*":
                return operand1 * operand2;
            case "/":
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Invalid operator: " +
operator);
        }
    }

    public static void main(String[] args) {
        String postfixExpression = "5 2 + 8 3 - * 4 /";
        int result = evaluatePostfix(postfixExpression);
        System.out.println("Result: " + result); // Output: 6
    }
}
```

4. Implement the clone( ) method for the ArrayStack class.

```
@Override
public ArrayStack<E> clone() {
    try {
        ArrayStack<E> clone = (ArrayStack<E>) super.clone();
        clone.stackArray = Arrays.copyOf(stackArray, stackArray.length);
```

```
            return clone;
        } catch (CloneNotSupportedException e) {
            throw new InternalError(e);
        }
    }
```

5. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value

```
import java.util.Scanner;
import java.util.Stack;

public class PostfixEvaluator {
    public static int evaluatePostfix(String expression) {
        Stack<Integer> stack = new Stack<>();

        String[] tokens = expression.split(" ");

        for (String token : tokens) {
            if (isOperator(token)) {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                int result = performOperation(token, operand1, operand2);
                stack.push(result);
            } else {
                int operand = Integer.parseInt(token);
                stack.push(operand);
            }
        }

        return stack.pop();
    }

    private static boolean isOperator(String token) {
        return token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/");
    }

    private static int performOperation(String operator, int operand1, int operand2) {
        switch (operator) {
            case "+":
                return operand1 + operand2;
            case "-":
```

الاسم : زهرة خالد

تقنية معلومات عربي

```
                return operand1 - operand2;
            case "*":
                return operand1 * operand2;
            case "/":
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Invalid operator: " + operator);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an expression in postfix notation: ");
        String postfixExpression = scanner.nextLine();
        int result = evaluatePostfix(postfixExpression);
        System.out.println("Result: " + result);
    }
}
```