Data Structure Lab7 : Stack 2022-2023

Topics

1. Create Stack Interface
2. Create Stack Using Array
3. Create Stack Using Linked Lists
4. Implement Basic Methods of Stack
    - isEmpty()
    - size()
    - top()
    - push(E e)
    - pop()

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 void transfer(Stack<Integer> S, Stack<Integer> T) {
    // استخدام مكدس مؤقت لتخزين عناصر S بترتيب عكسي.
    Stack<Integer> temp = new Stack<>();
    while (!S.isEmpty()) {
    temp.push(S.pop());
    }

    // نقل عناصر من المكدس المؤقت إلى T.
    while (!temp.isEmpty()) {
    T.push(temp.pop());
    }
    }

// مثال على الاستخدام
        Stack<Integer> S = new Stack<>();
    S.push(1);
    S.push(2);
    S.push(3);
```

```
        Stack<Integer> T = new Stack<>();
        transfer(S, T);

        while (!T.isEmpty()) {
        System.out.println(T.pop());
        }
```

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

```
    public static void popAll(Stack<Integer> S) {
        // قاعدة التوقف: إذا كان المكدس فارغًا، فلا يوجد شيء لإزالته
            if (S.isEmpty()) {
        return;
        }

        //إزالة العنصر العلوي للمكدس
            S.pop();

        // استدعاء التكرار لإزالة العناصر المتبقية
            popAll(S);
        }

    // مثال على الاستخدام
            Stack<Integer> S = new Stack<>();
        S.push(1);
        S.push(2);
        S.push(3);

        popAll(S);

        while (!S.isEmpty()) {
        System.out.println(S.pop()); // لن يطبع شيئًا لأنه تم إزالة جميع العناصر
            }
```

3.

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 double evaluate(String expression) {
        // Stack to hold intermediate values
        Stack<Double> stack = new Stack<>();

        // Tokenize the expression, assuming valid input
        String[] tokens = expression.split(" ");

        // Process each token
        for (String token : tokens) {
            try {
                double operand = Double.parseDouble(token); // Convert operand to double
                stack.push(operand);
            } catch (NumberFormatException e) {
                // Operator: evaluate using popped operands
                double operand2 = stack.pop();
                double operand1 = stack.pop();
                double result;
                switch (token) {
                    case "+":
                        result = operand1 + operand2;
                        break;
                    case "-":
                        result = operand1 - operand2;
                        break;
                    case "*":
                        result = operand1 * operand2;
                        break;
                    case "/":
                        // Handle division by zero
                        if (operand2 == 0) {
                            throw new ArithmeticException("Division by zero");
                        }
                        result = operand1 / operand2;
                        break;
                    default:
```

```
                    throw new IllegalArgumentException("Invalid operator: " + token);
                }
                stack.push(result);
            }
        }

        // Ensure only one numeric value remains
        if (stack.size() != 1) {
            throw new IllegalArgumentException("Invalid expression: extra operands");
        }

        return stack.pop(); // Final result
    }

    public static void main(String[] args) {
        String expression = "5 2 + 8 3 - * 4 /";
        double result = evaluate(expression);
        System.out.println("The postfix expression evaluates to: " + result);
    }
}
```

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

```
public class ArrayStack<T> implements Cloneable {

    private T[] data;
    private int top;

    // other methods and functionality of your ArrayStack class

    @Override
    public ArrayStack<T> clone() throws CloneNotSupportedException {
        // Check if object can be cloned
        if (!super.cloneSupported()) {
            throw new CloneNotSupportedException("ArrayStack cannot be cloned");
        }

        // Create a new ArrayStack object
        ArrayStack<T> clone = new ArrayStack<>();

        // Allocate a new array to avoid shallow copying
        clone.data = (T[]) new Object[data.length];

        // Copy elements from original data to clone's data
        for (int i = 0; i <= top; i++) {
            clone.data[i] = data[i];
        }

        // Set clone's top index
        clone.top = top;
```

```
          // Return the cloned object
          return clone;
        }
      }
```

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

```
import java.util.Stack;

public class PostfixEvaluator {

    public static double evaluate(String expression) {
        Stack<Double> stack = new Stack<>();

        for (String token : expression.split(" ")) {
            try {
                double operand = Double.parseDouble(token);
                stack.push(operand);
            } catch (NumberFormatException e) {
                // Operator processing
                if (stack.size() < 2) {
                    throw new IllegalArgumentException("Insufficient operands for operator: " + token);
                }
                double operand2 = stack.pop();
                double operand1 = stack.pop();
                double result;
                switch (token) {
                    case "+":
                        result = operand1 + operand2;
                        break;
                    case "-":
                        result = operand1 - operand2;
                        break;
                    case "*":
                        result = operand1 * operand2;
                        break;
                    case "/":
                        if (operand2 == 0) {
                            throw new ArithmeticException("Division by zero");
                        }
                        result = operand1 / operand2;
                        break;
                    default:
                        throw new IllegalArgumentException("Invalid operator: " + token);
                }
                stack.push(result);
            }
        }

        if (stack.size() != 1) {
            throw new IllegalArgumentException("Invalid expression: extra operands");
        }

        return stack.pop();
    }

    public static void main(String[] args) {
        String expression = "5 2 + 8 3 - * 4 /";
        double result = evaluate(expression);
        System.out.println("The postfix expression evaluates to: " + result);
    }
}
```