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.

```
import java.util.Stack;

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

    public static void main(String[] args) {
        Stack<Integer> S = new Stack<>();
        Stack<Integer> T = new Stack<>();
```

```
        S.push(1);
        S.push(2);
        S.push(3);

        transfer(S, T);

        System.out.println("Stack T after transfer: " + T); // Output: [3, 2, 1]
    }
}
```

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

```
public class RecursiveRemove {
    public static <E> void removeAll(Stack<E> stack) {
        if (!stack.isEmpty()) {
            stack.pop(); // Remove top element
            removeAll(stack); // Recursive call
        }
    }

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);

        removeAll(stack);

        System.out.println("Stack after removal: " + stack); // Output: []
    }
}
```

Data Structure Lab7 : Stack 2022-2023

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

        for (char ch : expression.toCharArray()) {
            if (Character.isDigit(ch)) {
                stack.push(ch - '0'); // Convert char to int
            } else {
                int b = stack.pop();
                int a = stack.pop();
                switch (ch) {
                    case '+': stack.push(a + b); break;
                    case '-': stack.push(a - b); break;
                    case '*': stack.push(a * b); break;
                    case '/': stack.push(a / b); break;
                }
            }
        }
```

```
        return stack.pop();
    }

    public static void main(String[] args) {
        String expression = "52+83-*4/";
        System.out.println("Result: " + evaluatePostfix(expression)); // Output:
3
    }
}
```

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

```
public class ArrayStack<E> implements Cloneable {
    private E[] data;
    private int top = -1;

    public ArrayStack(int capacity) {
        data = (E[]) new Object[capacity];
    }

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

    public void push(E e) {
        if (top == data.length - 1) throw new IllegalStateException("Stack is
full");
        data[++top] = e;
    }

    public E pop() {
        if (isEmpty()) return null;
        E result = data[top];
        data[top--] = null;
```

```
        return result;
    }

    @Override
    public ArrayStack<E> clone() {
        try {
            ArrayStack<E> cloned = (ArrayStack<E>) super.clone();
            cloned.data = data.clone(); // Deep copy
            return cloned;
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }
}
```

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

        for (char ch : expression.toCharArray()) {
            if (Character.isDigit(ch)) {
                stack.push(ch - '0'); // Convert char to int
            } else {
                int b = stack.pop();
                int a = stack.pop();
                switch (ch) {
                    case '+': stack.push(a + b); break;
                    case '-': stack.push(a - b); break;
                    case '*': stack.push(a * b); break;
```

```
                    case '/': stack.push(a / b); break;
                }
            }
        }

        return stack.pop();
    }

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