Stack and Queue Hands On

1. Stack Implementation Using Array

```
package usr.hands_on;

import usr.collections.StackADT;

public class Qn_1 {
    public static void main(String[] args) {
        StackADT<Integer> stack = new StackADT<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);
        System.out.println("removed element: "+stack.pop());
        System.out.println("top element in stack: " + stack.getTop());
        System.out.println("is the stack is empty? " +stack.isEmpty());
        stack.display();
    }
}
```

Output

```
removed element: 50
top element in stack: 40
is the stack is empty? false
Stack ADT:
        40 <- Peek
        30
        20
        10
```

2. Stack using two queues

```
package usr.hands_on;

import usr.collections.QueueADT;

public class Qn_2 {
    static QueueADT<Integer> q = new QueueADT<>();
    static QueueADT<Integer> q2 = new QueueADT<>();

    public static void push(int val) {
        if (q.isEmpty())
            q.enqueue(val);
        else {
            int qSize = q.size();
            for(int i = 0; i < qSize; i++) {
                q2.enqueue(q.dequeue());
            }
            q.enqueue(val);
            for(int i = 0; i < qSize; i++) {
                q.enqueue(q2.dequeue());
            }
        }
    }
```

```
    public static int pop() {
        int val = q.dequeue();
        return val;
    }

    public static void main(String[] args) {
        push(10);
        push(20);
        push(30);
        push(40);
        push(50);
        System.out.println(pop());
        System.out.println("Stack: " + q.toString());
    }
}
```

Output

```
50
Stack: [40, 30, 20]
```

3. Queue Reversal

```
package usr.hands_on;

import usr.collections.QueueADT;

public class Qn_3 {
    public static void reverse(QueueADT<Integer> q) {
        if (q.isEmpty()) return;
        int val = q.dequeue();
        reverse(q);
        q.enqueue(val);
    }

    public static void main(String[] args) {
        QueueADT<Integer> q = new QueueADT<>();
        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.enqueue(40);
        q.enqueue(50);
        reverse(q);
        System.out.println("Reversed Queue: " + q.toString());
    }
}
```

Output

```
Reversed Queue: [50, 40, 30, 20, 10]
```

---

4. Remove All Adjacent Duplicates in String

```
package usr.hands_on;

import usr.collections.StackADT;
```

```java
import java.util.Scanner;

public class Qn_4 {
    public static void main(String[] args) {
        StackADT<Character> stack = new StackADT<>();
        Scanner sc = new Scanner(System.in);

        String str = sc.nextLine().trim();

        for(char c : str.toCharArray()){
            if (stack.getTop() == null){
                stack.push(c);
            } else if (stack.getTop() == c) {
                stack.pop();
            } else {
                stack.push(c);
            }
        }
        System.out.println(stack);


    }
}
```

Output

```
Input: abbaca
Output: ca
```

---

5. Backspace String Compare

```java
package usr.hands_on;

import usr.collections.StackADT;

import java.util.Scanner;

public class Qn_5 {
    public static void main(String[] args) {
        StackADT<Character> stack = new StackADT<>();
        StackADT<Character> stack2 = new StackADT<>();
        Scanner sc = new Scanner(System.in);

        String input1 = sc.nextLine();
        String input2 = sc.nextLine();

        for (char c : input1.toCharArray()) {
            if (stack.isEmpty() && c != '#') {
                stack.push(c);
            } else if (c == '#') {
                stack.pop();
            } else {
                stack.push(c);
            }
        }

        for (char c : input2.toCharArray()) {
            if (stack2.isEmpty() && c != '#') {
                stack2.push(c);
            } else if (c == '#') {
                stack2.pop();
            } else {
```

```
                stack2.push(c);
            }
        }

        Object[] arr = stack.toArray();
        Object[] arr2 = stack2.toArray();

        boolean flag = false;
        if (stack.getTop() != null || stack.getTop() != null)
            if (arr[0] == arr2[0] && stack.getTop() == stack2.getTop()) {
                flag = true;
            }
        System.out.println(flag);
    }
}
```

Output

```
Input: ab#c, ad#c
Output: true
```

---

6. Special Stack

```
package usr.hands_on;

import usr.collections.StackADT;

public class Qn_6 {
    public static void main(String[] args) {
        StackADT<Integer> stack = new StackADT<>();

        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(1);
        stack.push(40);
        stack.push(5);
        stack.push(50);

        System.out.println(stack.getMin());
    }
}
```

Output

```
1
```

---

7. Queue using two Stacks

```
package usr.hands_on;

import usr.collections.StackADT;


public class Qn_7 {
    static StackADT<Integer> s1 = new StackADT<>();
    static StackADT<Integer> s2 = new StackADT<>();
```

```
    public static void enqueue(int val) {
        if (s1.isEmpty())
            s1.push(val);
        else {
            int s1size = s1.size();
            for( int i=0; i<s1size; i++ ) {
                s2.push(s1.pop());
            }
            s1.push(val);
            for ( int i=0; i<s1size; i++ ) {
                s1.push(s2.pop());
            }
        }
    }

    public static int dequeue() {
        int val = s1.pop();
        return val;
    }

    public static void main(String[] args) {

        enqueue(10);
        enqueue(20);
        enqueue(30);
        enqueue(40);
        enqueue(50);
        System.out.println(dequeue());
        System.out.println(s1);

    }
}
```

Output

```
10
[50, 40, 30, 20]
```

---

8. Postfix To Prefix

```
package usr.hands_on;

import usr.collections.StackADT;

import java.util.Scanner;

public class Qn_8 {
    static boolean isOperator(char x) {
        switch (x) {
            case '+':
            case '-':
            case '/':
            case '*':
                return true;
        }
        return false;
    }
    static String postToPre(String post_exp) {
        StackADT<String> s = new StackADT<>();
        int length = post_exp.length();
        for (int i = 0; i < length; i++) {
```

```
            if (isOperator(post_exp.charAt(i))) {
                String op1 = s.peek();
                s.pop();
                String op2 = s.peek();
                s.pop();
                String temp = post_exp.charAt(i) + op2 + op1;
                s.push(temp);
            }
            else {
                s.push(post_exp.charAt(i) + "");
            }
        }

        StringBuilder ans = new StringBuilder();
        for (Object i : s.toArray())
            ans.append(i);
        return ans.toString();
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your postfix expression: ");
        String post_exp = sc.nextLine();
        System.out.println("Prefix : "
                + postToPre(post_exp));
    }
}
```

Output

```
Enter your postfix expression: ABCD^E-FGH+**^*
Prefix : ****+A^BC-^DE+FGH
```

Stack Class

```
package usr.collections;

import java.util.Arrays;
import java.util.Comparator;

public class StackADT<E extends Comparable<E>> {
    private E[] stack;
    private int top;
    int size = 10;

    @SuppressWarnings("unchecked")
    public StackADT() {
        stack = (E[]) new Comparable[10];
        top = -1;
    }

    @SuppressWarnings("unchecked")
    public StackADT(int size) {
        stack = (E[]) new Comparable[size];
        top = -1;
    }

    public void push(E element) {
        if (top == stack.length - 1) {
            // Resize the stack if it's full
            resizeStack();
        }
```

```
        stack[++top] = element;
    }

    public E pop() {
        if (isEmpty()) {
            return null;
        }
        return stack[top--];
    }

    public E peek() {
        if (isEmpty()) {
            return null;
        }
        return stack[top];
    }

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

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

    public void display() {
        System.out.println("Stack ADT:");
        for (int i = top; i >= 0; i--) {
            if (i == top) {
                System.out.println("    " + stack[i] + " <- Peek");
            } else {
                System.out.println("    " + stack[i]);
            }
        }
    }

    public void reverse() {
        int i = 0, j = top;
        while (i < j) {
            E temp = stack[i];
            stack[i] = stack[j];
            stack[j] = temp;
            i++;
            j--;
        }
    }

    public E getTop(){
        if (top == -1)
            return null;
        return stack[top];
    }

    public void sort() {
        Arrays.sort(stack, 0, top+1);
    }

    private void resizeStack() {
        stack = Arrays.copyOf(stack, stack.length * 2);
    }

    public int search(E data) {
        for(int i=0; i < top; i++ )
            if (stack[i] == data)
                return i;
        return -1;
    }
```

```
    public boolean contains(E data) {
        for(int i=0; i < top; i++ )
            if (stack[i] == data)
                return true;
        return false;
    }

    @Override
    public String toString() {
        if (isEmpty())
            return "Stack is empty";
        return Arrays.toString(Arrays.copyOfRange(stack,0,top+1));
    }

    public Object[] toArray() {
        if (isEmpty())
            return new Object[0];
        return Arrays.copyOfRange(stack,0,top+1);
    }

    public E getMin(){
        return Arrays.stream(Arrays.copyOfRange(stack,0,top+1)).min(Comparator.comparing(x -> x)).orElse(null);
    }
}
```

Queue Class

```
package usr.collections;

import java.util.Arrays;

public class QueueADT <E extends Comparable<E>> {
    private E[] queue;
    private int front, end;

    public QueueADT() {
        queue = (E[]) new Comparable[10];
        front = -1;
        end = -1;
    }

    public QueueADT(int capacity) {
        queue = (E[]) new Comparable[capacity];
        front = -1;
        end = -1;
    }

    public void enqueue(E e) {
        if (end == queue.length - 1) {
            // Queue is full, resize the array
            resizeQueue();
        }

        if (front == -1) {
            // Queue is empty
            front = 0;
        }

        queue[++end] = e;
    }

    public E dequeue() {
        if (front == -1) {
            // Queue is empty
```

```
            return null;
        }

        E item = queue[front];
        if (front == end) {
            // Only one element in the queue
            front = -1;
            end = -1;
        } else {
            front++;
        }

        return item;
    }

    public int size() {
        if (front == -1) {
            // Queue is empty
            return 0;
        }

        return end - front + 1;
    }

    private void resizeQueue() {
        E[] newQueue = (E[]) new Comparable[queue.length * 2];
        for (int i = front; i <= end; i++) {
            newQueue[i - front] = queue[i];
        }
        queue = newQueue;
        front = 0;
        end = end - front;
    }

    public void display() {
        if (front == -1) {
            // Queue is empty
            System.out.println("Queue is empty.");
            return;
        }

        System.out.print("Queue: ");
        for (int i = front; i <= end; i++) {
            System.out.print(queue[i] + " ");
        }
        System.out.println();
    }

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

    @Override
    public String toString() {
        return Arrays.toString(Arrays.copyOfRange(queue,front,end));
    }

}
```