public class ArrayStack <E> implements Stack<E> {  
 ///Q1  
 public static void transfer(int[] source, int[] target) {  
 int sourceSize = source.length;  
 int targetSize = target.length;  
 int targetIndex = targetSize - 1;  
  
 for (int i = sourceSize - 1; i >= 0; i--) {  
 target[targetIndex] = source[i];  
 targetIndex--;  
 }  
 }  
  
 ///Q2  
 public static void removeAll(int[] stack) {  
 int stackSize = stack.length;  
  
 if (stackSize > 0) {  
 int[] newStack = new int[stackSize - 1];  
 System.arraycopy(stack, 0, newStack, 0, stackSize - 1);//يزيل اخر عنصر في كل مرة  
 removeAll(newStack);  
 }  
 }  
 ////Q3 AND Q5 postfix  
  
 ///Q4  
 public ArrayStack<E> clone() {  
 try {  
 return (ArrayStack<E>) super.clone();  
 } catch (CloneNotSupportedException e) {  
 throw new AssertionError(e); // This should not happen  
 }  
 }  
  
 private E data[];  
 private int t = -1;  
  
 // public ArrayStack(int cap)  
// {  
// data= (E[]) new Object[cap];  
// }  
 @Override  
 public int size() {  
 return t + 1;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return t == -1;  
 }  
  
 @Override  
 public void push(E element) {  
 if (size() == data.length) throw new  
 IllegalStateException("stack is full");  
 data[++t] = element;  
 }  
  
 @Override  
 public E pop() {  
 if (isEmpty()) return null;  
 E deleted = data[t];  
 data[t] = null;  
 t--;  
 return deleted;  
 }  
  
 @Override  
 public E top() {  
 if (isEmpty()) return null;  
 return data[t];  
 }  
 }

//حل السؤال الثالث والخامس  
import java.util.Stack;  
  
public class PostfixEvaluator {  
 //Q3  
 public static int evaluatePostfix(String postfixExpression) {  
 Stack<Integer> stack = new Stack<>();  
  
 for (int i = 0; i < postfixExpression.length(); i++) {  
 char ch = postfixExpression.charAt(i);  
  
 if (Character.isDigit(ch)) {  
 // If the character is a digit, push it onto the stack  
 stack.push(ch - '0');  
 } else if (ch != ' ') {  
 // If the character is an operator, perform the corresponding operation  
 int operand2 = stack.pop();  
 int operand1 = stack.pop();  
  
 switch (ch) {  
 case '+':  
 stack.push(operand1 + operand2);  
 break;  
 case '-':  
 stack.push(operand1 - operand2);  
 break;  
 case '\*':  
 stack.push(operand1 \* operand2);  
 break;  
 case '/':  
 stack.push(operand1 / operand2);  
 break;  
 }  
 }  
 }  
  
 // The final result is the only element left on the stack  
 return stack.pop();  
 }  
  
// public static void main(String[] args) {  
// String postfixExpression = "5 2 + 8 3 - \* 4 /";  
// int result = evaluatePostfix(postfixExpression);  
//  
// System.out.println("Result: " + result);  
// }  
 //Q5  
 public static int evaluatePostfixExpression(String expression) {  
 Stack<Integer> stack = new Stack<>();  
  
 for (int i = 0; i < expression.length(); i++) {  
 char ch = expression.charAt(i);  
  
 if (Character.isDigit(ch)) {  
 stack.push(Character.getNumericValue(ch));  
 } else {  
 int operand2 = stack.pop();  
 int operand1 = stack.pop();  
  
 int result;  
  
 switch (ch) {  
 case '+':  
 result = operand1 + operand2;  
 break;  
 case '-':  
 result = operand1 - operand2;  
 break;  
 case '\*':  
 result = operand1 \* operand2;  
 break;  
 case '/':  
 result = operand1 / operand2;  
 break;  
 default:  
 throw new IllegalArgumentException("Invalid operator: " + ch);  
 }  
  
 stack.push(result);  
 }  
 }  
  
 return stack.pop();  
 }  
  
// public static void main(String[] args) {  
// String postfixExpression = "52+83-\*4/";  
//  
// int result = evaluatePostfixExpression(postfixExpression);  
// System.out.println("Result: " + result);  
// }  
}