public class Lab7 {  
 /\*  
  
  
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.  
 قم بتنفيذ طريقة نقل التوقيع (S، T) التي تنقل جميع العناصر من المكدس S إلى المكدس T،  
 بحيث يكون العنصر الذي يبدأ في الجزء العلوي من S هو أول من يتم إدراجه في T، والعنصر الموجود في الأسفل ينتهي S في الجزء العلوي من T.  
  
الداله t بنقل جمبع العنصر  
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.  
2. أعط طريقة عودية لإزالة كافة العناصر من المكدس.  
تقوم بحذف جميع العناصر منr  
public static <E> void removeElements(Stack<E> stack) {  
 if (!stack.isEmpty()) {  
 stack.pop();  
 removeElements(stack);  
 }  
}  
  
  
  
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.  
3. تدوين Postfix هو طريقة لا لبس فيها لكتابة تعبير حسابي بدون أقواس. يتم تعريفه بحيث إذا كان  
"(exp1)op(exp2)" تعبيرًا عاديًا بين قوسين بالكامل وتشغيله هو op، فإن إصدار postfix من هذا هو "pexp1 pexp2 op"، حيث pexp1 هو إصدار postfix  
من exp1 وpexp2 هو نسخة postfix من exp2. إصدار postfix لرقم واحد أو متغير هو مجرد هذا الرقم أو المتغير. لذا، على سبيل المثال،  
 النسخة اللاحقة من "((5 + 2) ∗ (8 − 3))/4" هي "5 2 + 8 3 − ∗ 4 /". وصف طريقة غير متكررة لتقييم تعبير في تدوين postfix.  
  
تقيم دالهp اذا كان عمليه حسابي استخراج اثنين ووضعه في ستاك واذا رقم تحويل لدبل  
public static double evaluatePostfixExpression(String expression) {  
 Stack<Double> stack = new Stack<>();  
  
 String[] tokens = expression.split(" ");  
 for (String token : tokens) {  
  
 if (isNumber(token)) {  
 stack.push(Double.parseDouble(token));  
 } else {  
 double operand2 = stack.pop();  
 double operand1 = stack.pop();  
 double result = evaluateOperator(token, operand1, operand2);  
 stack.push(result);  
 }  
 }  
4. Implement the clone( ) method for the ArrayStack class.  
. قم بتنفيذ طريقة الاستنساخ () لفئة ArrayStack  
 تحذير تعريف الدالهc لانشاء نسخه  
 @SuppressWarnings("unchecked")  
@Override  
public ArrayStack<E> clone() throws  
  
CloneNotSupportedException {  
 ArrayStack<E> clone = (ArrayStack<E>) super.clone();  
 clone.data = data.clone();  
 return clone;  
}  
  
  
  
5. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value  
 قم بتنفيذ برنامج يمكنه إدخال تعبير بتدوين لاحق (راجع التمرين C-6.19) وإخراج قيمته  
 تقوم بتقيم تقسيم الى رموز صغيره فرديه وتقيم كل رمز  
public static double evaluatePostfixExpression(Strin  
  
g expression) {  
 Stack<Double> stack = new Stack<>();  
  
 String[] tokens = expression.split(" ");  
 for (String token : tokens) {  
 if (isNumber(token)) {  
 stack.push(Double.parseDouble(token));  
 } else {  
 double operand2 = stack.pop();  
 double operand1 =  
  
stack.pop();  
 double result = evaluateOperator(token, operand1, operand2);  
 stack.push(result);  
 }  
 }  
  
 return stack.pop();  
}  
  
private static boolean isNumber(String token) {  
 return token.matches("-?\\d+(\\.\\d+)?");  
  
}  
  
private static double evaluateOperator(String operator, double operand1, double 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) {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.print("Enter an expression in postfix notation: ");  
 String expression = scanner.nextLine();  
 double result = evaluatePostfixExpression(expression);  
 System.out.println("Result: " + result);  
}  
  
 \*/  
}