/\*

Arthur Flores

masc0809

\*/

import data\_structures.\*;

import java.util.StringTokenizer;

public class ExpressionEvaluator{

//isOperator is a method that check to see if the //element passed through the method is an operator

//The return value for this method is a true/false stating //wether its an operator or not. There are no error //conditions here.

private boolean isOperator(String operator){

return operator.equals("+") ||

operator.equals("-") ||

operator.equals("\*") ||

operator.equals("/") ||

operator.equals("^");

}

//precedence is a method that places a integer value as //the precedence of a operator. The value

//that is returned is the integer of precedence.There are //no error conditions here.

private int precedence(String element){

if(element.equals("+") || element.equals("-"))

return 1;

else if(element.equals("\*") || element.equals("/"))

return 2;

else if(element.equals("^"))

return 3;

else

return 0;

}

//isNumber is a method that checks to see if the element //passed in is a number or not.The return

//value is a boolean of true/false defining if the element //is a number.The error condition here

// is caught using the try/catch block but returns a false //if an error has occured.

private boolean isNumber(String element){

try{

double d=Double.parseDouble(element);

return true;

}

catch(NumberFormatException e){

return false;

}

}

//doOperation is a method that does the actual math for //the two number that were popped of the

//queue. The return value is the answer to the math that //was done. The only error that is handled

//in this method is if a number is divided by 0, in which //cases brings an error message to the screen.

private String doOperation(double data1, double data2, String operator){

Double result;

if(operator.equals("+") )

result = data1 + data2;

else if(operator.equals("-"))

result = data1 - data2;

else if(operator.equals("\*"))

result = data1 \* data2;

else if(operator.equals("^")){

result=Math.pow(data1,data2);

}

else{

if(data2 == 0)

throw new RuntimeException();

result = data1/data2;

}

String endResult= result+"";

return endResult;

}

//processInput is called by calculator and turns the //string s from infix to postifix and then does

//the math. This method returns answer which is the final //result from the calculations. There are

//many error caught here using parenthesis and numbers to //see if they are in the correct spot. The

//main error condition is when you try to convert elements //dequeued from the queue and turning it into a number.

public String processInput(String s){

if(s==null)

return "0";

try{

Stack<String> stack = new Stack<String>();

Queue<String> queue = new Queue<String>();

StringTokenizer tokenizer = new StringTokenizer(s);

String previous = "";

stack.makeEmpty();

queue.makeEmpty();

while (tokenizer.hasMoreTokens()) {

String element = tokenizer.nextToken();

if(element.equals("(")){

if(isNumber(previous))

throw new RuntimeException();

stack.push(element);

previous=element;

}

else if(element.equals(")")){

while(!stack.peek().equals("("))

queue.enqueue(stack.pop());

stack.pop();

previous=element;

}

else if(isOperator(element)){

if(stack.isEmpty() && queue.isEmpty())

throw new RuntimeException();

while(!stack.isEmpty() && !stack.peek().equals("(") &&

precedence(stack.peek())>=precedence(element))

queue.enqueue(stack.pop());

stack.push(element);

previous=element;

}

else{ // it is a number

if(previous.equals(")"))

throw new RuntimeException();

queue.enqueue(element);

previous=element;

}

}

if(isOperator(previous) || previous.equals("("))

throw new RuntimeException();

while(!stack.isEmpty())

queue.enqueue(stack.pop());

// The queue now has the postfix expression

//An algorithm for evaluating a postfix expression:

while(!queue.isEmpty()){

if(isNumber(queue.peek()))

stack.push(queue.dequeue());

else{

Double data2= Double.parseDouble(stack.pop());

Double data1= Double.parseDouble(stack.pop());

String finalResult= doOperation(data1,data2,queue.dequeue()); stack.push(finalResult);

}

}

String answer=stack.pop();

if(!stack.isEmpty() || !queue.isEmpty())

throw new RuntimeException();

return answer;

}

catch(Exception e){

return "ERROR";

}

}

}