**import** java.math.BigDecimal;

**import** java.math.RoundingMode;

**import** java.util.ArrayList;

/\*\*

\* inner class evaluate expression that contain methods for evaluate expressions

\*

\*

\*/

**public** **class** EvaluateExpression **extends** Calculator {

ArrayList<String> contents;

String item;

EvaluateExpression check;

/\*\*

\* this method for evaluate first brackets because brackets is more priority

\*

\* **@param** <E>

\* **@param** e

\* **@return**

\*/

**public** <E> E brackets(E e) {

check = **new** EvaluateExpression();

String s = e.toString();

**while** (s.contains(Character.*toString*('(')) || s.contains(Character.*toString*(')'))) {

**for** (**int** i = 0; i < s.length(); i++) {

**if** (s.charAt(i) == ')') {

**for** (**int** j = i; j >= 0; j--) {

**if** (s.charAt(j) == '(') {

String in = s.substring(j + 1, i);

in = check.recognize(in);

s = s.substring(0, j) + in + s.substring(i + 1);

j = i = 0;

}

}

}

}

}

s = check.recognize(s);

**return** (E) s;

}

/\*\*

\* method for dismount expression and rearrange expression to be prepared

\* for evaluating

\*

\* **@param** <E>

\* **@param** e

\* **@return**

\*/

**public** <E> E recognize(E e) {

PutIt putIt = **new** PutIt();

contents = **new** ArrayList<String>();

String s = e.toString();

item = "";

**for** (**int** i = s.length() - 1; i >= 0; i--) {

**if** (Character.*isDigit*(s.charAt(i))) {

item = s.charAt(i) + item;

**if** (i == 0) {

putIt.put();

}

}

**else** {

**if** (s.charAt(i) == '.') {

item = s.charAt(i) + item;

}

**else** **if** (s.charAt(i) == '-' && (i == 0 || (!Character.*isDigit*(s.charAt(i - 1))))) {

item = s.charAt(i) + item;

putIt.put();

}

**else** {

putIt.put();

item += s.charAt(i);

putIt.put();

**if** (s.charAt(i) == '|') {

item += " ";

putIt.put();

}

}

}

}

contents = putIt.result(contents, "@", "&");

contents = putIt.result(contents, "#", "!");

contents = putIt.result(contents, "^", "%");

contents = putIt.result(contents, "|", "$");

contents = putIt.result(contents, "\*", "/");

contents = putIt.result(contents, "+", "-");

**return** (E) contents.get(0);

}

/\*\*

\* inner class contains methods that filter expression and using math java

\* classes for evaluate expression

\*/

**public** **class** PutIt {

/\*\*

\* method that prepare final arithmetic statment

\*/

**public** **void** put() {

**if** (!item.equals("")) {

contents.add(0, item);

item = "";

}

}

/\*\*

\* method that we send to it array list contains arithmetic statment and

\* 2 operators and evaluate expressions of this operators and return

\* array list of after evaluate

\* **@param** arrayList

\* **@param** op1

\* **@param** op2

\* **@return**

\*/

**public** ArrayList<String> result(ArrayList<String> arrayList, String op1, String op2) {

**int** scale = 5;

String e = "";

BigDecimal result = **new** BigDecimal(0);

**for** (**int** c = 0; c < arrayList.size(); c++) {

**if** (arrayList.get(c).equals(op1) || arrayList.get(c).equals(op2)) {

**if** (arrayList.get(c).equals("@")) {

result = **new** BigDecimal(Math.*sin*((Double.*parseDouble*(arrayList.get(c + 1)))\*(Math.***PI***/180)));

}

**else** **if** (arrayList.get(c).equals("&")) {

result = **new** BigDecimal(Math.*cos*((Double.*parseDouble*(arrayList.get(c + 1)))\*(Math.***PI***/180)));

}

**else** **if** (arrayList.get(c).equals("#")) {

result = **new** BigDecimal(Math.*tan*((Double.*parseDouble*(arrayList.get(c + 1)))\*(Math.***PI***/180)));

}

**else** **if** (arrayList.get(c).equals("|")) {

result = **new** BigDecimal(Math.*sqrt*(Double.*parseDouble*(arrayList.get(c + 1))));

}

**else** **if** (arrayList.get(c).equals("\*")) {

result = **new** BigDecimal(arrayList.get(c - 1)).multiply(**new** BigDecimal(arrayList.get(c + 1)));

}

**else** **if** (arrayList.get(c).equals("/")) {

**if** (arrayList.get(c + 1).equals("0")) {//divide by zero

e = "error expression";

arrayList.clear();

arrayList.add(e);

**return** arrayList;

} **else** **if** (arrayList.get(c + 1).equals(".")) {

e = "error expression";

arrayList.clear();

arrayList.add(e);

**return** arrayList;

}

result = **new** BigDecimal(arrayList.get(c - 1)).divide(**new** BigDecimal(arrayList.get(c + 1)), scale,

BigDecimal.***ROUND\_DOWN***);

}

**else** **if** (arrayList.get(c).equals("+")) {

result = **new** BigDecimal(arrayList.get(c - 1)).add(**new** BigDecimal(arrayList.get(c + 1)));

}

**else** **if** (arrayList.get(c).equals("-")) {

result = **new** BigDecimal(arrayList.get(c - 1)).subtract(**new** BigDecimal(arrayList.get(c + 1)));

}

**else** **if** (arrayList.get(c).equals("%")) {

**if** (arrayList.get(c + 1).equals("0")) {//divide by zero

e = "error expression";

arrayList.clear();

arrayList.add(e);

**return** arrayList;

}

**else**

result = **new** BigDecimal(arrayList.get(c - 1)).remainder(**new** BigDecimal(arrayList.get(c + 1)));

}

**try** {

**if**(arrayList.get(c).equals("@") || arrayList.get(c).equals("#")||arrayList.get(c).equals("&")){

arrayList.set(c, (result.setScale(scale, RoundingMode.***HALF\_DOWN***).

stripTrailingZeros().toPlainString()));

arrayList.remove(c + 1);

}

**else**{

arrayList.set(c, (result.setScale(scale, RoundingMode.***HALF\_DOWN***).

stripTrailingZeros().toPlainString()));

arrayList.remove(c + 1);

arrayList.remove(c - 1);

}

}

**catch** (Exception ignored) {

Calculator cal=**new** Calculator();

cal.showExpression="";

cal.hiddenExpression="";

cal.textarea.setText("error");

}

}

**else** {

**continue**;

}

c = 0;

}

**return** arrayList;

}

}

}