Aziza Mankenova, 2018400387

Assignment 2

1. **Problem description:**

The given problem involves writing a Java program to evaluate arithmetic expressions using four lines of input which will be provided by the user. The order of precedence for the operations should be as follows, parentheses first, then multiplication-division and then addition-subtraction. User will enter three different variables in the first three lines and some expression, which might include variables entered before, in the fourth line. The program should evaluate all of the arithmetic operations between numbers and variables and print the final result.

1. **Problem Solution**

In order to write the program 10 static methods were used. Also, to write the code of the program while loops, for loops, if-else conditionals and switch conditionals were implemented.

The formatLine(**String** line1) method is used to format one of the first three lines which has a variable declaration and initialization, using as a parameter the line that should be formatted. It returns a line ,which has one space after the variable type, one space before and after a variable and equal sign.

The formatInput(**String** line4) method is used to format the fourth line of input, which contains the expression that should be evaluated, using as a parameter the line that should be formatted. It uses one string parameter which should be formatted. It returns a line in which all of the numbers and operators are separated by whitespaces. Implementing this method makes the further evaluations easier.

The replaceVaribale(**String** line1, **String** line) method is used to replace all the variables in the fourth line by its numerical values, using two string variables as parameters. One of the strings contains the variable type, name and number and the other parameter is the one that might contain that variable. It returns the line in which a specific variable is replaced.

The numberBefore(**String** line, **int** indexOfOperator) method is used to find the number before the specified operator. It takes a line as a string parameter and indexOfOperator as an integer parameter. It returns a string containing the number before the given operator.

The numberAfter(**String** line, **int** indexOfOperator) method is used to find the number after the specified operator. It takes a line as a string parameter and indexOfOperator as an integer parameter. It returns a string containing the number after the given operator.

The evaluateOperation(**String** line, **int** indexOfOperator, **char** operator) method is used to evaluate the specified arithmetical operation inside the line. It takes a line as a string parameter, indexOfOperator as an integer parameter, and an operator as a char parameter. As a result it returns a new string in which a certain operation(includes two numbers and an operator between them) is replaced by its numeric value.

The returnIntResult(**int** num1, **int** num2, **char** operator) method is used to return an integer result of the operation between two integer numbers. It takes two integer numbers and a char operator as parameters and performs an operation on this two numbers depending on the type of the operator. Since both of the numbers are integers the returned result is also an integer.

The returnDoubleResult(**double** num1, **double** num2, **char** operator) method is used to return a double result of the operation between two double numbers. It takes two double numbers and a char operator as parameters and performs an operation on this two numbers depending on the type of the operator. Since both of the numbers are double the returned result is also a double.

The checkForArithmeticOper(**String** line, **char** a, **char** b) method checks and performs arithmetic operations of either (multiplication and division) or (summation and subtraction). It takes a line as a string parameter, and two char (could be “+” and “-“ or “\*” and “/”, because the precedence of “\*” and “/” is the same, the same is applicable to “+” and “-“ ) parameters. It returns a string in which the specific operations are performed.

The parentheses(**String** line) method is used to find parentheses and perform operations inside of them. It takes a line as a string parameter and returns a string in which the operations in the parentheses are performed.

1. **Implementation**

**package** assignment2;

**import** java.util.Scanner;

**public** **class** AM2018400387 {

**public** **static** **void** main(String[] args) {

Scanner console = **new** Scanner(System.***in***);

String line10 = console.nextLine();

String line20 = console.nextLine();

String line30 = console.nextLine();

String line4 = console.nextLine();

console.close();

String line1 = *formatLine*(line10);//formatting the line using the formatLine method

String line2 = *formatLine*(line20);//formatting the line using the formatLine method

String line3 = *formatLine*(line30);//formatting the line using the formatLine method

String line = *formatInput*(line4);// formatted input using formatInput method

line = *replaceVaribale*(line1,line);//replacing variable by its a numeric value in the fourth line(if present) using method replaceVaribale

line = *replaceVaribale*(line2,line);//replacing variable by its a numeric value in the fourth line(if present) using method replaceVaribale

line = *replaceVaribale*(line3,line);//replacing variable by its a numeric value in the fourth line(if present) using method replaceVaribale

//checking the fourth line for all parentheses and performing operations inside of them using parentheses method

line = *parentheses*(line);

//checking the fourth line for multiplication and division operations using the method checkForArithmeticOper

line = *checkForArithmeticOper*(line,'\*','/');

//checking the fourth line for summation and subtraction operations using the method checkForArithmeticOper

line = *checkForArithmeticOper*(line,'+','-');

// printing the result in the console without the the semicolon

System.***out***.println(line.replace(';', ' '));

}

//This method is used to find parentheses and perform operations inside of them

**public** **static** String parentheses(String line) {

//using while loop to find all the parentheses

**while**(line.contains(")")){

//taking the first inner parentheses and storing these expression in the new variable toCal

String s = line.substring(0, line.indexOf(")")+1) ;

String toCalc = s.substring(s.lastIndexOf("("));

//checking for multiplication and subtraction inside the parentheses, and performing all the operations that exist by the order of precedence

String expression = *checkForArithmeticOper*(toCalc,'\*','/');

expression = *checkForArithmeticOper*(expression,'+','-');

//the result inside the brackets is stored in the expression variable

expression = expression.substring(expression.indexOf("(")+1, expression.length()-1);

//replacing the parentheses with some arithmetic operations by the result of these operation(s)

line = line.replace(toCalc, expression);

//formatting the line

line = *formatInput*(line);

}

**return** line;

}

//this method is used to replace all the variables in the fourth line by its numerical values.

**public** **static** String replaceVaribale(String line1, String line) {

//the name variable is used to store the name of the declared variable

String name = line1.substring(line1.indexOf(" ") + 1, line1.indexOf("=") - 1);

//the number variable stores the number assigned to the variable

String number = line1.substring(line1.indexOf("=") + 2, line1.indexOf(";"));

//using if else statement to determine if the it is an integer or double value

**if**(line1.substring(0,3).equals("int")) // if it is an integer

{

**if**(line.contains(name)) {

line = line.replace(name, number);

}

}**else** // it must be a double variable

{

**if**(number.contains(".")) {//if the double contained a dot

//replace the variable in the last line by its numeric value

line = line.replace(name, number);

}**else** {// the double did not have a dot

//add a dot to the number

number = number + ".";

//replace the variable in the last line by its numeric value

line = line.replace(name, number);

}

}

**return** line;

}

//This method is used to format one of the first three lines which has a variable declaration and initialization

**public** **static** String formatLine(String line1) {

Scanner exp = **new** Scanner(line1);

String variableLine = "";

//writing all tokens without spaces using a while loop

**while**(exp.hasNext()) {

variableLine += exp.next();

}

exp.close();

variableLine = variableLine.toLowerCase();

//replacing the "int", "double" and "=" by their spaced versions

variableLine = variableLine.replace("int", "int ");

variableLine = variableLine.replace("double", "double ");

variableLine = variableLine.replace("=", " = ");

**return** variableLine;

}

//this method is used to format the fourth line of input

**public** **static** String formatInput(String line4) {

Scanner exp = **new** Scanner(line4);

String expression = "";

// writing all tokens without spaces using a while loop

**while**(exp.hasNext()) {

expression += exp.next();

}

exp.close();

String line = "";

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

**char** a = expression.charAt(i);

//checking every character of the expression and adding spaces before and after the characters,which are not numbers

**if**((a == '+') || (a == '-') || (a == '/') || (a == '\*') || (a=='(') || (a==')')) {

line += " " + a + " ";

}

**else** {

line += a;

}

}

**return** line;

}

//this method finds the number before the specified operator

**public** **static** String numberBefore(String line, **int** indexOfOperator) {

//the num variable stores the substring before that operator

String num = line.substring(0, indexOfOperator-1);

//if num has space inside of it, it would cut the string, and return the substring without spaces, which is our number

**if**(num.contains(" ")) {

num = num.substring(num.lastIndexOf(" ")+1);

}

**return** num;

}

//this method finds the number after the specified operator

**public** **static** String numberAfter(String line, **int** indexOfOperator) {

//the num variable stores the substring after that operator

String num = line.substring(indexOfOperator + 2);

**if**(num.contains(" ")) {//if num has space inside of it, it would take the string, and return the substring without spaces, which is our number

num = num.substring(0, num.indexOf(" ") );

}**else** {// num does not have any spaces but has a semicolon which is cut off the string by the substring method, and we get our number

num = num.substring(0, num.indexOf(";") );

}

**return** num;

}

//this method is used to evaluate the specified arithmetical operation inside the line

**public** **static** String evaluateOperation(String line, **int** indexOfOperator, **char** operator) {

String result = "";

//storing the number before the certain operator in the number1 variable

String number1 = *numberBefore*(line,line.indexOf(operator));

//storing the number after the certain operator in the number2 variable

String number2 = *numberAfter*(line, line.indexOf(operator));

//if number1 and number2 did not have dots

**if**(!number1.contains(".") && !number2.contains(".")){

//get integer numeric value of number1 and store it in the num1 variable

**int** num1 = Integer.*parseInt*(number1);

//get integer numeric value of number2 and store it in the num2 variable

**int** num2 = Integer.*parseInt*(number2);

//store the result of the operation between num1 and num2 as a string in the result variable

result = Integer.*toString*(*returnIntResult*(num1,num2,operator));

}

**else**{//one of the number1 or number2 or both of them might have dots

//get double numeric value of number1 and store it in the num1 variable

**double** num1 = Double.*parseDouble*(number1);

//get double numeric value of number2 and store it in the num2 variable

**double** num2 = Double.*parseDouble*(number2);

//store the result of the operation between num1 and num2 as a string in the result variable

result = Double.*toString*(*returnDoubleResult*(num1, num2, operator));

}

//replacing the number1, operator and number 2 with the result of the operation between them

line = line.replace(number1 + " " + operator + " " + number2, result);

**return** line;

}

// This method is used to return an integer result of the operation between two integer numbers

**public** **static** **int** returnIntResult(**int** num1, **int** num2, **char** operator) {

**int** result = 0;

//using switch to perform the operation between two numbers depending on the type of the operator

**switch**(operator) {

**case** '+' :

result = num1 + num2;

**break**;

**case** '-' :

result = num1 - num2;

**break**;

**case** '\*' :

result = num1 \* num2;

**break**;

**case** '/' :

result = num1 / num2;

**break**;

}

**return** result;

}

// This method is used to return a double result of the operation between two double numbers

**public** **static** **double** returnDoubleResult(**double** num1, **double** num2, **char** operator) {

**double** result = 0;

//using switch to perform the operation between two numbers depending on the type of the operator

**switch**(operator) {

**case** '+' :

result = num1 + num2;

**break**;

**case** '-' :

result = num1 - num2;

**break**;

**case** '\*' :

result = num1 \* num2;

**break**;

**case** '/' :

result = num1 / num2;

**break**;

}

**return** result ;

}

//This method checks and performs arithmetic operations of either (multiplication and division) or (summation and subtraction)

**public** **static** String checkForArithmeticOper(String line, **char** a, **char** b) {

//using while loop to perform the operations while "a" and "b" operators exist

**while**( line.contains(""+a) || line.contains(""+b) ) {

//if the line has both of the operators

**if**( line.contains(""+a) && line.contains(""+b) ) {

//storing index of the first operator

**int** index1 = line.indexOf(a);

//storing index of the second operator

**int** index2 = line.indexOf(b);

//while loop that checks while the the index1 is less than index2

**while**(index1 < index2) {

//perform the operation which corresponds to operator "a"

line = *evaluateOperation*(line, index1, a);

//if both of the operators exist

**if**(line.contains(""+a) && line.contains(""+b)) {

//assign new values to index1 and index2

index1 = line.indexOf(a);

index2 = line.indexOf(b);

**continue**;

}

**else** {//one of the operators or both of them does not exist

**break**;

}

}

//while loop that checks while the index1 is greater than index2

**while**(index1 > index2) {

//perform the operation which corresponds to operator "b"

line = *evaluateOperation*(line, line.indexOf(b), b);

//if both of the operators exist

**if**(line.contains(""+a) && line.contains(""+b)) {

//assign new values to index1 and index2

index1 = line.indexOf(a);

index2 = line.indexOf(b);

**continue**;

}

**else** {//one of the operators or both of them does not exist

**break**;

}

}

// if line has "a" operator, but not "b" operator

}**else** **if**(line.contains(""+a) && !(line.contains(""+b))){

//perform the operation which corresponds to operator "a"

line = *evaluateOperation*(line, line.indexOf(a), a);

}**else** {// line must have an operator "b", but not "a"

//perform the operation which corresponds to operator "b"

line = *evaluateOperation*(line, line.indexOf(b), b);

}

}

**return** line;

}

}

1. **Output of the program**

****



1. **Conclusion**

The program functions and gives the output it was expected to give. The program can evaluate all of the arithmetic operations between numbers and variables and print the final result. Overall, the given problem was solved.