Необходимо написать программу на языке программирования Java, реализующую функционал калькулятора по модели MVC (Model-View-Controller).

Функции которые должны присутствовать у калькулятора:

1. Производить расчеты уравнений без неизвестных. Пр. (7+8\*23/2,5)\*\*23.

Операции:

Сложение - +

Вычитание - "-"

Умножение - \*

Деление - /

Возведение в степень - \*\* или ^

Деление без остатка - //

Остаток от деления - %

Отделение частей уравнения с помощью скобок - ()

2. Сохранять уравнения. введенные ранее и результаты и иметь возможность их просмотра после перезапуска.

3. Сохранять файл с результатами посчитанных ранее уравнений в отдельный файл.

В случае если пользователь не указывает имя и путь для сохранения - указываем где храниться текущий файл с историей;

Если пользователь указал имя файла и расширение (txt, log, md) - сохраняем отдельный файл с этим именем в папке где запущена программа и указываем полный путь;

Если пользователь указал лишь путь где следует сохранить файл - сохраняем файл по введенному пути с именем log.log

Если пользователь указал абсолютный путь с именем файла - сохраняем файл в папке которая указана пользователем с именем указанным пользователем.

4. Выбирать уравнения которые необходимо сохранить и сохранять их в отдельный файл.

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

public class CalculatorMVC {

public static void main(String[] args) {

CalculatorModel model = new CalculatorModel();

CalculatorView view = new CalculatorView();

CalculatorController controller = new CalculatorController(model, view);

controller.run();

}

}

class CalculatorModel {

private List<String> history;

private String filePath = "calculator\_history.log";

public CalculatorModel() {

history = loadHistory();

}

public double evaluateExpression(String expression) {

try {

return new ExpressionEvaluator().eval(expression);

} catch (Exception e) {

throw new IllegalArgumentException("Invalid expression: " + expression);

}

}

public void addToHistory(String expression, double result) {

history.add(expression + " = " + result);

}

public List<String> getHistory() {

return history;

}

public String getFilePath() {

return filePath;

}

public void setFilePath(String filePath) {

this.filePath = filePath;

}

public void saveHistory() {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

for (String item : history) {

writer.write(item);

writer.newLine();

}

} catch (IOException e) {

System.err.println("Error saving history: " + e.getMessage());

}

}

private List<String> loadHistory() {

List<String> loadedHistory = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

String line;

while ((line = reader.readLine()) != null) {

loadedHistory.add(line);

}

} catch (IOException e) {

// Ignore if file not found (first run)

}

return loadedHistory;

}

}

class CalculatorView {

private BufferedReader reader;

public CalculatorView() {

reader = new BufferedReader(new InputStreamReader(System.in));

}

public void displayMessage(String message) {

System.out.println(message);

}

public String getExpression() {

displayMessage("Enter an expression (or 'h' for history, 's' to save, 'c' to change save path, 'q' to quit):");

try {

return reader.readLine();

} catch (IOException e) {

throw new RuntimeException("Error reading input", e);

}

}

public void displayHistory(List<String> history) {

if (history.isEmpty()) {

displayMessage("History is empty.");

} else {

for (String item : history) {

displayMessage(item);

}

}

}

public void displayResult(double result) {

displayMessage("Result: " + result);

}

public String getFilePath() {

displayMessage("Enter a file path to save to (or leave empty to use default):");

try {

return reader.readLine();

} catch (IOException e) {

throw new RuntimeException("Error reading input", e);

}

}

}

class CalculatorController {

private CalculatorModel model;

private CalculatorView view;

public CalculatorController(CalculatorModel model, CalculatorView view) {

this.model = model;

this.view = view;

}

public void run() {

view.displayMessage("Calculator started. Default history file: " + model.getFilePath());

while (true) {

String input = view.getExpression();

if (input.equalsIgnoreCase("q")) {

break;

} else if (input.equalsIgnoreCase("h")) {

view.displayHistory(model.getHistory());

} else if (input.equalsIgnoreCase("s")) {

model.saveHistory();

view.displayMessage("History saved to: " + model.getFilePath());

} else if (input.equalsIgnoreCase("c")) {

String newPath = view.getFilePath();

updateFilePath(newPath);

view.displayMessage("History file path changed to: " + model.getFilePath());

} else {

try {

double result = model.evaluateExpression(input);

view.displayResult(result);

model.addToHistory(input, result);

} catch (IllegalArgumentException e) {

view.displayMessage(e.getMessage());

}

}

}

model.saveHistory(); // Auto-save on exit

view.displayMessage("Exiting calculator.");

}

private void updateFilePath(String newPath) {

if (newPath.isEmpty()) {

// Use default file path

} else if (newPath.contains(File.separator)) {

// Path with directory specified

model.setFilePath(newPath.endsWith(".log") ? newPath : newPath + File.separator + "log.log");

} else {

// Only filename specified, save in current directory

model.setFilePath(System.getProperty("user.dir") + File.separator + newPath);

}

}

}

// Simple expression evaluator (supports +, -, \*, /, %, \*\*, //, and parentheses)

class ExpressionEvaluator {

private int pos = -1, ch;

public double eval(String str) {

this.pos = -1;

this.ch = -1;

return parse(str);

}

private void nextChar(String str) {

ch = (++pos < str.length()) ? str.charAt(pos) : -1;

}

private boolean eat(int charToEat, String str) {

while (ch == ' ') {

nextChar(str);

}

if (ch == charToEat) {

nextChar(str);

return true;

}

return false;

}

private double parse(String str) {

nextChar(str);

double x = parseExpression(str);

return x;

}

private double parseExpression(String str) {

double x = parseTerm(str);

while (true) {

if (eat('+', str)) {

x += parseTerm(str);

} else if (eat('-', str)) {

x -= parseTerm(str);

} else {

return x;

}

}

}

private double parseTerm(String str) {

double x = parseFactor(str);

while (true) {

if (eat('\*', str)) {

x \*= parseFactor(str);

} else if (eat('/', str)) {

x /= parseFactor(str);

} else if (eat('%', str)) {

x %= parseFactor(str);

} else {

return x;

}

}

}

private double parseFactor(String str) {

if (eat('+', str)) {

return parseFactor(str);

}

if (eat('-', str)) {

return -parseFactor(str);

}

double x;

int startPos = this.pos;

if (eat('(', str)) {

x = parseExpression(str);

eat(')', str);

} else if ((ch >= '0' && ch <= '9') || ch == '.') {

while ((ch >= '0' && ch <= '9') || ch == '.') {

nextChar(str);

}

x = Double.parseDouble(str.substring(startPos, this.pos));

} else {

throw new RuntimeException("Unexpected: " + (char) ch);

}

if (eat('^', str) || eat('\*', str, str)) {

return Math.pow(x, parseFactor(str));

}

return x;

}

private boolean eat(char charToEat, String str, String str2) {

if (pos < str.length() - 1 && str.charAt(pos) == charToEat && str.charAt(pos + 1) == '\*') {

pos += 2;

ch = pos < str.length() ? str.charAt(pos) : -1;

return true;

}

return false;

}

}