Nomes: Alan Henrique Jahnel, Ismael Bortoluzzi

O programa aceita diversas operações, entre elas: +, -, *, /, sqrt(), ^, sin(). cos(), tan(), e() e euler().

Para usar o número de euler, basta digitar e(). Existe ainda um atalho para o euler elevado a um expoente, como e^2, e^3 ou e^x, que é o euler(). Por exemplo: e^2 = euler(2). O número de euler também pode ser elevado a um expoente deste jeito (e())^x.

Importante: Não esqueça de sempre usar parênteses para envolver toda a expressão que deseja elevar a um expoente, por exemplo: (sin(x))^2.

Para rodar o CalculadoraIntegrais.jar, basta ter o java instalado, abrir a pasta onde está o programa no cmd, e digitar:

```
java -jar  CalculadoraIntegrais.jar
```

O trabalho foi feito em java, logo o código fonte está dividido em 3 classes:

CLASSE Main :

```
import java.util.Scanner;
import java.util.Locale;

public class Main {
   public static void main(String[] args) {

      Locale.setDefault(Locale.US);
      Scanner sc = new Scanner(System.in);
      double a, b, n;
      String funcao;
```

```
        System.out.print("Digite a função que deseja para calcular a integral: ");
        funcao = sc.nextLine();
        System.out.print("Digite o início do Intervalo: ");
        a = sc.nextDouble();
        System.out.print("Digite o fim do Intervalo: ");
        b = sc.nextDouble();
        System.out.print("Digite quantidade de faixas: ");
        n = sc.nextDouble();

        Riemann soma = new Riemann(a, b, n, funcao);
        System.out.println(soma.getIntegralAsString());

        sc.close();
    }

}
```

CLASSE Riemann :

```
import java.text.DecimalFormat;

public class Riemann {

    private double a;
    private double b;
    private double n;
    private String funcao;

    public Riemann(double a, double b, double n, String funcao) {

        this.a = a;
        this.b = b;
        this.n = (a+b)/n;
        this.funcao = funcao;

    }

    public double[] calculateIntegral() {

        double[] integrais = new double[2];
        integrais[0] = integrais[1] = 0;
```

```
        DecimalFormat df = new DecimalFormat("#");
        df.setMaximumFractionDigits(20);

        for(double i = this.a+n; i<=b; i+=n) {

            integrais[0] += n * Operacoes.eval(funcao.replaceAll("x", df.format(i)));
        }

        for(double i = this.a; i<b; i+=n) {

            integrais[1] += n * Operacoes.eval(funcao.replaceAll("x", df.format(i)));
        }

        integrais[0] = Math.abs(integrais[0]);
        integrais[1] = Math.abs(integrais[1]);

        return integrais;
    }

    public String getIntegralAsString() {

        double[] vect = calculateIntegral();
        return Double.toString(vect[0]) + " < A < " + Double.toString(vect[1]);
    }

}
```

CLASSE Operacoes :

```
public final class Operacoes {

    public static double eval(final String str) {
        return new Object() {
            int pos = -1, ch;

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

            boolean eat(int charToEat) {
                while (ch == ' ') nextChar();
                if (ch == charToEat) {
```

```
            nextChar();
            return true;
        }
        return false;
    }

    double parse() {
        nextChar();
        double x = parseExpression();
        if (pos < str.length()) throw new RuntimeException("Unexpected: " +
(char)ch);
        return x;
    }

    // Grammar:
    // expression = term | expression `+` term | expression `-` term
    // term = factor | term `*` factor | term `/` factor
    // factor = `+` factor | `-` factor | `(` expression `)`
    //        | number | functionName factor | factor `^` factor

    double parseExpression() {
        double x = parseTerm();
        for (;;) {
            if      (eat('+')) x += parseTerm(); // addition
            else if (eat('-')) x -= parseTerm(); // subtraction
            else return x;
        }
    }

    double parseTerm() {
        double x = parseFactor();
        for (;;) {
            if      (eat('*')) x *= parseFactor(); // multiplication
            else if (eat('/')) x /= parseFactor(); // division
            else if (eat('%')) x %= parseFactor(); // module
            else return x;
        }
    }

    double parseFactor() {
        if (eat('+')) return parseFactor(); // unary plus
        if (eat('-')) return -parseFactor(); // unary minus
```

```
            double x;
            int startPos = this.pos;
            if (eat('(')) { // parentheses
               if(eat(')')){
                  x = 0;
               }else{
                  x = parseExpression();
                  eat(')');
               }
            }else if ((ch >= '0' && ch <= '9') || ch == '.') { // numbers
               while ((ch >= '0' && ch <= '9') || ch == '.') nextChar();
               x = Double.parseDouble(str.substring(startPos, this.pos));
            } else if (ch >= 'a' && ch <= 'z') { // functions
               while (ch >= 'a' && ch <= 'z') nextChar();
               String func = str.substring(startPos, this.pos);
               x = parseFactor();
               if (func.equals("sqrt")) x = Math.sqrt(x);
               else if (func.equals("sin")) x = Math.sin(Math.toRadians(x));
               else if (func.equals("cos")) x = Math.cos(Math.toRadians(x));
               else if (func.equals("tan")) x = Math.tan(Math.toRadians(x));
               else if (func.equals("euler")) x = Math.exp(x);
               else if (func.equals("e")) x = Math.exp(1);
               else throw new RuntimeException("Unknown function: " + func);
            } else {
               throw new RuntimeException("Unexpected: " + (char)ch);
            }

            if (eat('^')) x = Math.pow(x, parseFactor()); // exponentiation

            return x;
         }
      }.parse();
   }

}
```