/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Bianca Lima Santos 10346811 \*\*/

/\*\* \*\*/

/\*\* 28/05/2017 \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class NewtonRaphson{

// depósitos realizados mais com saldo final

static double[] depositos = new double[11];

// datas de depósitos realizados e saldo final

static int[] datas = new int[11];

/\* método para calculo da função, retornando o seu resultado \*/

static double funcao(double j) {

double resultado = 0;

/\* resultado é a soma de deposito x 1+j elevado a ultima data de deposito - a data da posição atual, tudo isso em um for que incrementa

\* valor obtido na variavel resultado até que 1 seja menor que 10, ou seja, o array depositos seja percorrido por completo

\*/

for (int i=0; i<10; i++) {

resultado += depositos[i] \* Math.pow(1 + j, datas[10] - datas[i]);

}

resultado -= depositos[10];

return resultado;

}

/\* método para calculo da função derivada, retornando o seu resultado \*/

static double funcderiv(double j) {

double resultado = 0;

/\* resultado aqui é a soma da operação: ultima data de deposito - a data da posição atual x

\* deposito x 1+j elevado a ultima data de deposito - a data da posição atual -1,

\* (mesma operação realizada anteriormente com aplicação da regra do tombo usada para obter a derivada primeira de uma função)

\* tudo isso dentro de um for que incrementa valor obtido na variavel resultado até que 1 seja menor que 10, ou seja,

\* o array depositos seja percorrido por completo

\*/

for (int i=0; i<10; i++) {

resultado += (datas[10] - datas[i]) \* depositos[i] \* Math.pow(1+j, (datas[10] - datas[i])-1);

}

resultado -= depositos[10];

return resultado;

}

/\* método calcula os juros segundo Newton-Raphson \*/

static double newton(double epsilon) {

// tratamento para quando epsilon for menor ou igual a zero ou maior ou igual a 1

if(epsilon <= 0 || epsilon >= 1) {

return -1;

}

// juros antes do calculo

double jk\_antes = 0;

// juros pos o calculo

double jk\_pos = 0.5;

do {

// atribuição do valor de juros antes = o valor de juros depois

jk\_antes = jk\_pos;

// juros apos será o valor do juros antes - o resultado da função dividido pelo resultado da derivada primeira

jk\_pos = jk\_antes - funcao(jk\_antes)/funcderiv(jk\_antes);

} // isso será executado até que o valor absoluto do juros apos o calculo - o juros antes do calco seja maior ou igual ao valor de epsilon definido

while (Math.abs(jk\_pos - jk\_antes) >= epsilon);

// codigo para que o resultado seja apresentado apenas com três casas decimais

int jk\_curto = (int)(jk\_pos \* (1/epsilon));

return (double)(jk\_curto/(1/epsilon));

}

/\* main, usado para testar a aplicação alimentando os vetores depositos e datas e chamando método newton \*/

public static void main(String[] args) {

// depositos realizados

depositos[0] = 100;

depositos[1] = 2000;

depositos[2] = 1000;

depositos[3] = 100;

depositos[4] = 900;

depositos[5] = 120;

depositos[6] = 100;

depositos[7] = 220;

depositos[8] = 0;

depositos[9] = 0;

// saldo total

depositos[10] = 5000;

// datas dos depositos

datas[0] = 3;

datas[1] = 4;

datas[2] = 5;

datas[3] = 8;

datas[4] = 9;

datas[5] = 10;

datas[6] = 10;

datas[7] = 10;

datas[8] = 11;

datas[9] = 12;

// data do ultimo deposito realizado

datas[10] = 12;

// imprime o resultado final do programa

System.out.println("Resultado: " + newton(0.001));

}

}