/\*\*

\* Computes the periodical payment necessary to re-pay a given loan.

\*/

public class LoanCalc {

static double epsilon = 0.001; // The computation tolerance (estimation error)

static int iterationCounter; // Monitors the efficiency of the calculation

/\*\*

\* Gets the loan data and computes the periodical payment.

\* Expects to get three command-line arguments: sum of the loan (double),

\* interest rate (double, as a percentage), and number of payments (int).

\*/

public static void main(String[] args) {

// Gets the loan data

double loan = Double.parseDouble(args[0]);

//double loan = 100000;

double rate = Double.parseDouble(args[1]);

//double rate = 10;

int n = Integer.parseInt(args[2]);

//int n = 10;

System.out.println("Loan sum = " + loan + ", interest rate = " + rate + "%, periods = " + n);

// Computes the periodical payment using brute force search

System.out.print("Periodical payment, using brute force: ");

System.out.printf("%.2f", bruteForceSolver(loan, rate, n, epsilon));

System.out.println();

System.out.println("number of iterations: " + iterationCounter);

// Computes the periodical payment using bisection search

System.out.print("Periodical payment, using bi-section search: ");

System.out.printf("%.2f", bisectionSolver(loan, rate, n, epsilon));

System.out.println();

System.out.println("number of iterations: " + iterationCounter);

}

/\*\*

\* Uses a sequential search method ("brute force") to compute an approximation

\* of the periodical payment that will bring the ending balance of a loan close to 0.

\* Given: the sum of the loan, the periodical interest rate (as a percentage),

\* the number of periods (n), and epsilon, a tolerance level.

\*/

// Side effect: modifies the class variable iterationCounter.

public static double bruteForceSolver(double loan, double rate, int n, double epsilon) {

iterationCounter = 0;

double g = loan/n;

double x = endBalance(loan, rate, n, g);

while (x>0) {

g = g+epsilon;

x = endBalance(loan, rate, n, g);

iterationCounter ++;

}

// Replace the following statement with your code

return g;

}

/\*\*

\* Uses bisection search to compute an approximation of the periodical payment

\* that will bring the ending balance of a loan close to 0.

\* Given: the sum of theloan, the periodical interest rate (as a percentage),

\* the number of periods (n), and epsilon, a tolerance level.

\*/

// Side effect: modifies the class variable iterationCounter.

public static double bisectionSolver(double loan, double rate, int n, double epsilon) {

iterationCounter = 0;

double h = loan+epsilon;

double l = loan/n;

double g = (l+h)/2;

while ((h-l)>epsilon) {

double fl = endBalance(loan, rate, n, l);

double fg = endBalance(loan, rate, n, g);

if (fl\*fg>0){

l=g;

}

else{

h=g;

}

g = (l+h)/2;

iterationCounter ++;

}

// Replace the following statement with your code

return g;

}

/\*\*

\* Computes the ending balance of a loan, given the sum of the loan, the periodical

\* interest rate (as a percentage), the number of periods (n), and the periodical payment.

\*/

private static double endBalance(double loan, double rate, int n, double payment) {

while (n>0){

loan = (loan-payment)\*(1+rate/100);

n--;

// iterationCounter ++;

}

return loan;

}

}

/\*\* String processing exercise 1. \*/

public class lowercase {

public static void main(String[] args) {

String str = args[0];

System.out.println(lowerCase(str));

}

/\*\*

\* Returns a string which is identical to the original string,

\* except that all the upper-case letters are converted to lower-case letters.

\* Non-letter characters are left as is.

\*/

public static String lowerCase(String s) {

int a=0;

String result = "";

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

a = (int) s.charAt(i);

if( a<91 && a>64 ){

a+=32;

}

result += (char) a;

}

return result;

}

}