/\*\*

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

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

        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) {

        // Replace the following statement with your code

        double guess = loan / n;

        while (endBalance(loan, rate, n, guess) >= epsilon) {

            guess += epsilon;

            iterationCounter++;

        }

        return guess;

    }

    /\*\*

     \* 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) {

        // Replace the following statement with your code

        double lo = epsilon;

        double hi = loan;

        double guess;

        iterationCounter = 0; // Reset iteration counter

        while ((hi - lo) > epsilon) {

            guess = (lo + hi) / 2;

            if (endBalance(loan, rate, n, guess) \* endBalance(loan, rate, n, lo) > 0)//

                lo = guess;

            else

                hi = guess;

            iterationCounter++;

        }

        return (lo + hi) / 2;

    }

    /\*\*

     \* 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) {

        // Replace the following statement with your code

        double balance = loan;

        for (int i = 0; i < n; i++) {

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

        }

        return balance;

    }

}

/\*\* 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) {

        // Replace the following statement with your code

        String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        String lower = "abcdefghijklmnopqrstuvwxyz";

        String update = "";

        boolean check = false;

        int count;

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

            check = false;

            count = -1;

            for (int c = 0; c < upper.length(); c++) {

                if (s.charAt(i) == upper.charAt(c)) {

                    check = true;

                    count = c;

                }

            }

            if (check) {

                if (s.charAt(i) == upper.charAt(count))

                    update += lower.charAt(count);

                else

                    update += s.charAt(i);

            } else

                update += s.charAt(i);

        }

        return update;

    }

}

/\*\* String processing exercise 2. \*/

public class UniqueChars {

    public static void main(String[] args) {

        String str = args[0];

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

    }

    /\*\*

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

     \* except that all the duplicate characters are removed,

     \* unless they are space characters.

     \*/

    public static String uniqueChars(String s) {

        // Replace the following statement with your code

        String newS = "";

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

            char current = s.charAt(i);

            if (newS.indexOf(current) == -1 || current == ' ') {

                newS += current;

            }

        }

        return newS;

    }

}

/\*\*

 \* Prints the calendars of all the years in the 20th century.

 \*/

public class Calendar {

    // Starting the calendar on 1/1/1900

    static int dayOfMonth = 1;

    static int month = 1;

    static int year = 1900;

    static int dayOfWeek = 2; // 1.1.1900 was a Monday

    static int nDaysInMonth = 31; // Number of days in January

    static int specialSunday = 0; // number of special sundays

    /\*\*

     \* Prints the calendars of all the years in the 20th century. Also prints the

     \* number of Sundays that occured on the first day of the month during this

     \* period.

     \*/

    public static void main(String args[]) {

        int givenYear = Integer.parseInt(args[0]);

        // Advances the date and the day-of-the-week from 1/1/1900 till 31/12/1999,

        // inclusive.

        // Prints each date dd/mm/yyyy in a separate line. If the day is a Sunday,

        // prints "Sunday".

        // The following variable, used for debugging purposes, counts how many days

        // were advanced so far.

        int debugDaysCounter = 0;

        //// Write the necessary initialization code, and replace the condition

        //// of the while loop with the necessary condition

        while (year <= givenYear) {

            if (year == givenYear) {

                if (dayOfWeek == 1) {

                    if (dayOfMonth == 1)

                        specialSunday++;

                    System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");

                } else

                    System.out.println(dayOfMonth + "/" + month + "/" + year);

            }

            advance();

            debugDaysCounter++;

            //// If you want to stop the loop after n days, replace the condition of the

            //// if statement with the condition (debugDaysCounter == n)

        }

        //// Write the necessary ending code here

    }

    // Advances the date (day, month, year) and the day-of-the-week.

    // If the month changes, sets the number of days in this month.

    // Side effects: changes the static variables dayOfMonth, month, year,

    // dayOfWeek, nDaysInMonth.

    private static void advance() {

        // Replace this comment with your code

        dayOfWeek++;

        dayOfMonth++;

        if (dayOfMonth > nDaysInMonth(month, year))

        {

            month++;

            dayOfMonth = 1;

            if (month == 13) { // month ==13

                year++;

                month = 1;

            }

        }

        if (dayOfWeek > 7)

            dayOfWeek = 1;

    }

    // Returns true if the given year is a leap year, false otherwise.

    private static boolean isLeapYear(int year) {

        // Replace the following statement with your code

        boolean check;

        check = ((year % 400) == 0);

        check = check || (((year % 4) == 0) && ((year % 100) != 0));

        return check;

    }

    // Returns the number of days in the given month and year.

    // April, June, September, and November have 30 days each.

    // February has 28 days in a common year, and 29 days in a leap year.

    // All the other months have 31 days.

    private static int nDaysInMonth(int month, int year) {

        // Replace the following statement with your code

        if (isLeapYear(year)) {

            if (month == 2)

                return 29;

            else if ((month == 1) || (month == 3) || (month == 5) || (month == 7) || (month == 8) || (month == 10)

                    || (month == 12))

                return 31;

            else

                return 30;

        } else {

            if (month == 2)

                return 28;

            else if ((month == 1) || (month == 3) || (month == 5) || (month == 7) || (month == 8) || (month == 10)

                    || (month == 12))

                return 31;

            else

                return 30;

        }

    }

}