מטלה 3

/\*\*

\* 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;

}

}

/\*\* 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) {

String a="";

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

if(a.indexOf(s.charAt(i))==-1 || s.charAt(i)==' '){

a+=s.charAt(i);

}

}

return a;

}

}

/\*

\* Checks if a given year is a leap year or a common year,

\* and computes the number of days in a given month and a given year.

\*/

public class Calendar0 {

// Gets a year (command-line argument), and tests the functions isLeapYear and nDaysInMonth.

public static void main(String args[]) {

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

isLeapYearTest(year);

nDaysInMonthTest(year);

}

// Tests the isLeapYear function.

private static void isLeapYearTest(int year) {

String commonOrLeap = "common";

if (isLeapYear(year)) {

commonOrLeap = "leap";

}

System.out.println(year + " is a " + commonOrLeap + " year");

}

// Tests the nDaysInMonth function.

private static void nDaysInMonthTest(int year) {

for (int i =1; i<=12; i++){

System.out.println( "Month " +i+ " has "+ nDaysInMonth(i, year) +" days");

}

}

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

public static boolean isLeapYear(int year) {

if ( year%4==0 && year%100!=0 || year%400==0) {

return true;

}

return false;

}

// 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.

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

if (month== 4 || month == 6 || month == 9 || month == 11){

return 30;

}

else if (month == 2) {

if (isLeapYear(year)){

return 29;

}

else{

return 28;

}

}

else{

return 31;

}

}

}

/\*\*

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

\*/

public class Calendar1 {

// 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 sundayTheFirst = 0;

/\*\*

\* 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[]) {

// 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 < 2000) {

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

if (dayOfWeek==1){

//System.out.print(" Sunday");

if (dayOfMonth==1){

sundayTheFirst++;

}

}

//System.out.println();

advance();

debugDaysCounter++;

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

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

}

System.out.println("During the 20th century, "+ sundayTheFirst +" Sundays fell on the first day of the month");

//// 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() {

dayOfMonth++;

dayOfWeek++;

if (dayOfMonth>nDaysInMonth) {

month++;

nDaysInMonth = nDaysInMonth(month, year);

dayOfMonth = 1;

}

if (month > 12){

month = 1;

year ++;

}

if (dayOfWeek>7) {

dayOfWeek=1;

}

}

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

private static boolean isLeapYear(int year) {

if ( year%4==0 && year%100!=0 || year%400==0) {

return true;

}

return false;

}

// 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) {

if (month== 4 || month == 6 || month == 9 || month == 11){

return 30;

}

else if (month == 2) {

if (isLeapYear(year)){

return 29;

}

else{

return 28;

}

}

else{

return 31;

}

}

}

/\*\*

\* 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 sundayTheFirst = 0;

/\*\*

\* 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 year2print = 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 <=year2print) {

if (year==year2print) {

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

if (dayOfWeek==1){

System.out.print(" Sunday");

}

System.out.println();

}

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() {

dayOfMonth++;

dayOfWeek++;

if (dayOfMonth>nDaysInMonth) {

month++;

nDaysInMonth = nDaysInMonth(month, year);

dayOfMonth = 1;

}

if (month > 12){

month = 1;

year ++;

}

if (dayOfWeek>7) {

dayOfWeek=1;

}

}

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

private static boolean isLeapYear(int year) {

if ( year%4==0 && year%100!=0 || year%400==0) {

return true;

}

return false;

}

// 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) {

if (month== 4 || month == 6 || month == 9 || month == 11){

return 30;

}

else if (month == 2) {

if (isLeapYear(year)){

return 29;

}

else{

return 28;

}

}

else{

return 31;

}

}

}