

```

/**
 * 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 g = loan/n;
        iterationCounter=0;
        while(endBalance(loan, rate, n, g) >= epsilon) {
            g = g + epsilon;
            iterationCounter++;
        }
        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 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 bisectionSolver(double loan, double rate, int n, double epsilon) {
        // Replace the following statement with your code
        double min = 0;
    }
}

```

```

double max = loan;

double g = ((min + max) / 2);

iterationCounter=0;

while(max - min > epsilon){

    if(endBalance(loan, rate, n, min) * endBalance(loan, rate, n, g) < 0 ){

        max = g;

        g = (min + g) / 2;

    }

    else{

        min = g;

        g = (max + g) / 2;

    }

    iterationCounter++;

}

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) {
```

```
    // 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
        int length = s.length();
        String newString = "";
        for(int i = 0 ; i < length ; i++) {
            char ch = s.charAt(i);
            if (ch >= 'A' && ch <= 'Z'){
                newString = newString + (char)(ch + 32);
            }
            else{
                newString = newString + (char)(ch + 0);
            }
        }
        return newString;
    }
}

```

```

/** 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 = "";
        int length = s.length();

        for (int i = 0 ; i < length ; i++){
            char ch = s.charAt(i);
            if(newS.indexOf(ch) < 0 || ch == ' '){
                newS = newS + ch;
            }

        }
        return newS;
    }
}

```

```

/*
 * 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) {

        // Replace this comment with your code

        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) {  
    // Replace the following statement with your code  
  
    boolean leap = false;  
  
    if (year % 4 == 0 || (year % 100 == 0 && year % 400 == 0)){  
        leap = true;  
    }  
  
    return leap;  
}
```

// 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) {  
    // Replace the following statement with your code  
  
    int days = 0;  
  
    switch (month) {  
        case 1:  
            days = 31;  
            break;  
        case 2:  
            if(isLeapYear(year) == true){  
                days = 29;  
                break;  
            }  
            else{  
                days = 28;  
                break;  
            }  
    }  
}
```


case 3:

```
    days = 31;  
    break;
```

case 4:

```
    days = 30;  
    break;
```

case 5:

```
    days = 31;  
    break;
```

case 6:

```
    days = 30;  
    break;
```

case 7:

```
    days = 31;  
    break;
```

case 8:

```
    days = 31;  
    break;
```

case 9:

```
    days = 30;  
    break;
```

case 10:

```
    days = 31;  
    break;
```

case 11:

```
    days = 30;  
    break;
```

case 12:

```
    days = 31;  
    break;
```

```
    }  
    return days;  
}  
}
```

```

/**
 * 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

    /**
     * Prints the calendars of all the years in the 20th century. Also prints the
     * number of Sundays that occurred 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.

        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year <= 1999 && month <= 12) {
            //// Write the body of the while
            advance();
            break;
            //// 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
```

```
    int sundaycounter = 0;
```

```
    for(int year = 1900 ; year <= 1999 ; year++){
```

```
        for(int month = 1 ; month <= 12 ; month++){
```

```
            nDaysInMonth = nDaysInMonth(month , year);
```

```
            for(int i = 1; i <= nDaysInMonth ; i++){
```

```
                if(dayOfWeek == 1){
```

```
                    System.out.println(i + "/" + month + "/" + year + "
```

```
sunday");
```

```
                        if(i == 1){
```

```
                            sundaycounter++;
```

```
                        }
```

```
                    }
```

```
                else{
```

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

```
                }
```

```
            dayOfWeek++;
```

```
            if(dayOfWeek > 7){
```

```
                dayOfWeek = 1;
```

```
            }
```

```

        }
        dayOfMonth = 1;
    }
}

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

```

// 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 leap = false;

    if (year % 4 == 0 && (year % 100 == 0 && year % 400 == 0)){
        leap = true;
    }

    return leap;
}

```

// 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

    int days = 0;

    switch (month) {
        case 1:
            days = 31;
            break;

        case 2:

```

```
        if(isLeapYear(year) == true){
            days = 29;
            break;
        }
        else{
            days = 28;
            break;
        }
    case 3:
        days = 31;
        break;
    case 4:
        days = 30;
        break;
    case 5:
        days = 31;
        break;
    case 6:
        days = 30;
        break;
    case 7:
        days = 31;
        break;
    case 8:
        days = 31;
        break;
    case 9:
        days = 30;
        break;
    case 10:
        days = 31;
```

```
        break;
    case 11:
        days = 30;
        break;
    case 12:
        days = 31;
        break;

    }
    return days;
}
}
```

```

/**
 * 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


    /**
     * Prints the calendars of all the years in the 20th century. Also prints the
     * number of Sundays that occurred 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.


        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition

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

        //// Write the body of the while

        advance(year);


        //// 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(int year) {
```

```
    // Replace this comment with your code
```

```
    for(int y = 1900 ; y <= year ; y++){
```

```
        for(int month = 1 ; month <= 12 ; month++){
```

```
            nDaysInMonth = nDaysInMonth(month , y);
```

```
            for(int i = 1; i <= nDaysInMonth ; i++){
```

```
                dayOfWeek++;
```

```
                if(dayOfWeek > 7){
```

```
                    dayOfWeek = 1;
```

```
                }
```

```
            }
```

```
            dayOfMonth = 1;
```

```
        }
```

```
    }
```

```
    for(int month = 1 ; month <= 12 ; month++){
```

```
        nDaysInMonth = nDaysInMonth(month , year);
```

```
        for(int i = 1; i <= nDaysInMonth ; i++){
```

```
            if(dayOfWeek == 1){
```

```
                System.out.println(i + "/" + month + "/" + year + "
```

```
sunday");
```

```

        }

        else{
            System.out.println(i + "/" + month + "/" + year);
        }

        dayOfWeek++;
        if(dayOfWeek > 7){
            dayOfWeek = 1;
        }

    }

    dayOfMonth = 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 leap = false;

    if (year % 4 == 0 && (year % 100 == 0 && year % 400 == 0)){
        leap = true;
    }

    return leap;
}

```

// 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  
    int days = 0;  
    switch (month) {  
        case 1:  
            days = 31;  
            break;  
        case 2:  
            if(isLeapYear(year) == true){  
                days = 29;  
                break;  
            }  
            else{  
                days = 28;  
                break;  
            }  
        case 3:  
            days = 31;  
            break;  
        case 4:  
            days = 30;  
            break;  
        case 5:  
            days = 31;  
            break;  
        case 6:  
            days = 30;  
            break;  
        case 7:  
            days = 31;  
            break;  
    }  
}
```

case 8:

days = 31;

break;

case 9:

days = 30;

break;

case 10:

days = 31;

break;

case 11:

days = 30;

break;

case 12:

days = 31;

break;

}

return days;

}

}