```
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 l/oan (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);


    }

    // Side effect: modifies the class variable iterationCounter.
    public static double bruteForceSolver(double loan, double rate, int n,
double epsilon) {

        double loanKeep = loan;
        double increment = 0.001;
        double annualPay = (loan/n) + increment;
        iterationCounter = 0;

        while (loan > epsilon){
            loan = loanKeep;
            for (int i = 0; i<n; i++){
                loan = (loan - annualPay)*((rate/100)+1);
            }
```

```
            annualPay = annualPay + increment;
            iterationCounter++;
        }

        if (annualPay > loanKeep) {
            System.out.println("Faild to find a solution" + annualPay);

        }

        return annualPay;
    }

    // Side effect: modifies the class variable iterationCounter.
    public static double bisectionSolver(double loan, double rate, int n, double
epsilon) {
        double loanKeep = loan;
        double upperBound = loan;
        double lowerBound = 0;
        double annualPay = 0;
        iterationCounter = 0;

        while (upperBound - lowerBound >=epsilon) {
            annualPay = ((lowerBound+upperBound)/2);
            loan = loanKeep;

            if (endBalance(loan, rate, n, annualPay) > 0) {
                lowerBound = annualPay;

            }
            else {
                upperBound = annualPay;
            }

            iterationCounter++;
        }

        if(annualPay > loanKeep){
            System.out.println("Faild to find a solution" + annualPay);
        }

        return annualPay;
    }
```

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

        double leftOfLoan = loan;
        double temp =leftOfLoan;
        for(int i=0; i<n; i++){
              temp = leftOfLoan;
              leftOfLoan = (leftOfLoan-payment)*(1 + rate/100);

           }

           return leftOfLoan;

        }

    }
```

```
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 i = 0 ;
       String ans = "";
       while (i<s.length()) {
        char ch = s.charAt(i);
        if (ch >= 65 && ch <=90) {
            ans = ans +(char)(s.charAt(i)+32);

        }

        else if (ch == ' ' || ch>90 || ch<65){
            ans = ans + ch;
        }

        i++;

      }
       return ans;
    }
}
```

```
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 ans = "" + (char)(s.charAt(0));
       int n = 0;

       for (int i = 0; i<s.length(); i++){
           n=ans.indexOf(s.charAt(i));
           if (n>=0 && s.charAt(i) != ' ') {
            ans = ans;
           }
           else {
            ans = ans + s.charAt(i);
           }
    }
    return ans;
}
}
```

```
public class Calendar {

    // Starting the calendar on 1/1/1900
    static int dayOfMonth = 1;
    static int month = 1;
    static int dayOfWeek = 2;     // 1.1.1900 was a Monday
    static int nDaysInMonth = 31; // Number of days in January


    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);
        int yearCounter = 1900;

        int sumSunday1 = 0;

        while (yearCounter <= year) {
            for(int j = 1; j <= 12; j++){
            for (int i=1; i<=nDaysInMonth(month,yearCounter); i++) {
                if (dayOfWeek==1) {
                    dayOfWeek++;
                    if (yearCounter == year) {
                        System.out.println(i+"/" + month + "/" + year + " is Sunday");
                    }


                }
                else if (dayOfWeek==7){
                    dayOfWeek = 1;
                    if (yearCounter == year) {
                    System.out.println(i+"/" + month + "/" + year);
                    }

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

                }

            }
            month++;
            dayOfMonth = 1;



        }

        month= 1;
```

```
        yearCounter++;


        }

    }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        boolean isLeapYear;
        // Checks if the year is divisible by 400
            isLeapYear = ((year % 400) == 0);
        // Then checks if the year is divisible by 4 but not by 100
            isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) !=
0));

            return isLeapYear;

    }

    // 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) {
        int numOfDays;

        if(isLeapYear(year)){
            switch (month) {
                case 1: numOfDays = 31;
                        break;
                case 2: numOfDays = 29;
                        break;
                case 3: numOfDays = 31;
                        break;
                case 4: numOfDays = 30;
                        break;
                case 5: numOfDays = 31;
                        break;
                case 6: numOfDays = 30;
                        break;
                case 7: numOfDays = 31;
                        break;
                case 8: numOfDays = 31;
                        break;
                case 9: numOfDays = 30;
                        break;
                case 10: numOfDays = 31;
                        break;
                case 11: numOfDays = 30;
```

```
                break;
            case 12: numOfDays = 31;
                break;

            default:numOfDays = 0;
               break;

        }
    }
        else{
        switch (month) {
            case 1: numOfDays = 31;
                break;
            case 2: numOfDays = 28;
                break;
            case 3: numOfDays = 31;
                break;
            case 4: numOfDays = 30;
                break;
            case 5: numOfDays = 31;
                break;
            case 6: numOfDays = 30;
                break;
            case 7: numOfDays = 31;
                break;
            case 8: numOfDays = 31;
                break;
            case 9: numOfDays = 30;
                break;
            case 10: numOfDays = 31;
                break;
            case 11: numOfDays = 30;
                break;
            case 12: numOfDays = 31;
                break;

            default:numOfDays = 0;
               break;

        }
    }

    return numOfDays;

  }
}
```