```
/**
* 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) {
    double g = (loan / n);
    while(endBalance(loan, rate, n, g) > 0) {
        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 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) {
    double l = (loan / n);
    double h = loan;
    double g = (l + h) / 2;
    iterationCounter = 0;
    while ((h - l) > epsilon) {
        if(((endBalance(loan, rate, n, g)) * endBalance(loan, rate, n, l)) > 0) {
            l = g;
        } else {
```

```
            h = g;
        }
        g = (l + h) / 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) {
    double x = 0;
    for(int i = 0; i < n; i++){
        x = (loan - payment) * ((rate / 100) + 1);
        loan = x;
     }
    return x;
    }
}
```

```
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

    public static String lowerCase(String s) {
        String answer = "";
        for(int i=0; i<s.length(); i++) {
            if((s.charAt(i) >= 65) && (s.charAt(i) <= 90)){
                int asci = s.charAt(i);
                asci += 32;
                char charResult = (char) asci;
                answer += charResult;
            } else {
                answer += s.charAt(i);
            }
        }
        return answer;
    }
}
```

```
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    public static String uniqueChars(String s) {
        String answer = "";
        answer += s.charAt(0);
        int x = s.length();
        for(int i = 1; i < x; i++){
            boolean noDuplicate = true;
            for(int j = (i-1); (j >= 0) && noDuplicate; j--){
                if (s.charAt(i) == ' ') {
                    noDuplicate = true;
                } else if (s.charAt(j) == s.charAt(i)) {
                    noDuplicate = false;
                }
            }
            if (noDuplicate) {
                answer += s.charAt(i);
            }
        }
        return answer;
    }
}
```

```
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



    public static void main(String args[]) {
       int givenYear = Integer.parseInt(args[0]);
        while (year < givenYear) {
            advance();
        }
        while (year == givenYear){
            if(dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
            } else {
                System.out.println(dayOfMonth + "/" + month + "/" + year);
            }
            advance();
        }
    }


    // Advances the date and day-of-the-week.
    // Side effects: changes dayOfMonth, month, year, dayOfWeek, nDaysInMonth.
     private static void advance() {
        dayOfWeek = (dayOfWeek % 7) + 1; // Update day of the week
        dayOfMonth++;
        if(dayOfMonth > nDaysInMonth(month, year)) {
            dayOfMonth = 1;
            month++;
```

```
        if(month > 12) {
            month = 1;
            year++;
        }
    }
 }

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    if ((year % 4) != 0) {
        return false;
    } else {
        if((year % 100 == 0) && (year % 400!= 0)) {
            return false;
        } else {
            return true;
        }
    }
}

// Returns the number of days in the given month and year.
private static int nDaysInMonth(int month, int year) {
    int days;
    if ((month == 4) || (month == 6) || (month == 9) || (month == 11)) {
        days = 30;
    } else if (month == 2) {
        if(isLeapYear(year)) {
            days = 29;
        } else {
            days = 28;
        }
    } else {
        days = 31;
```

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