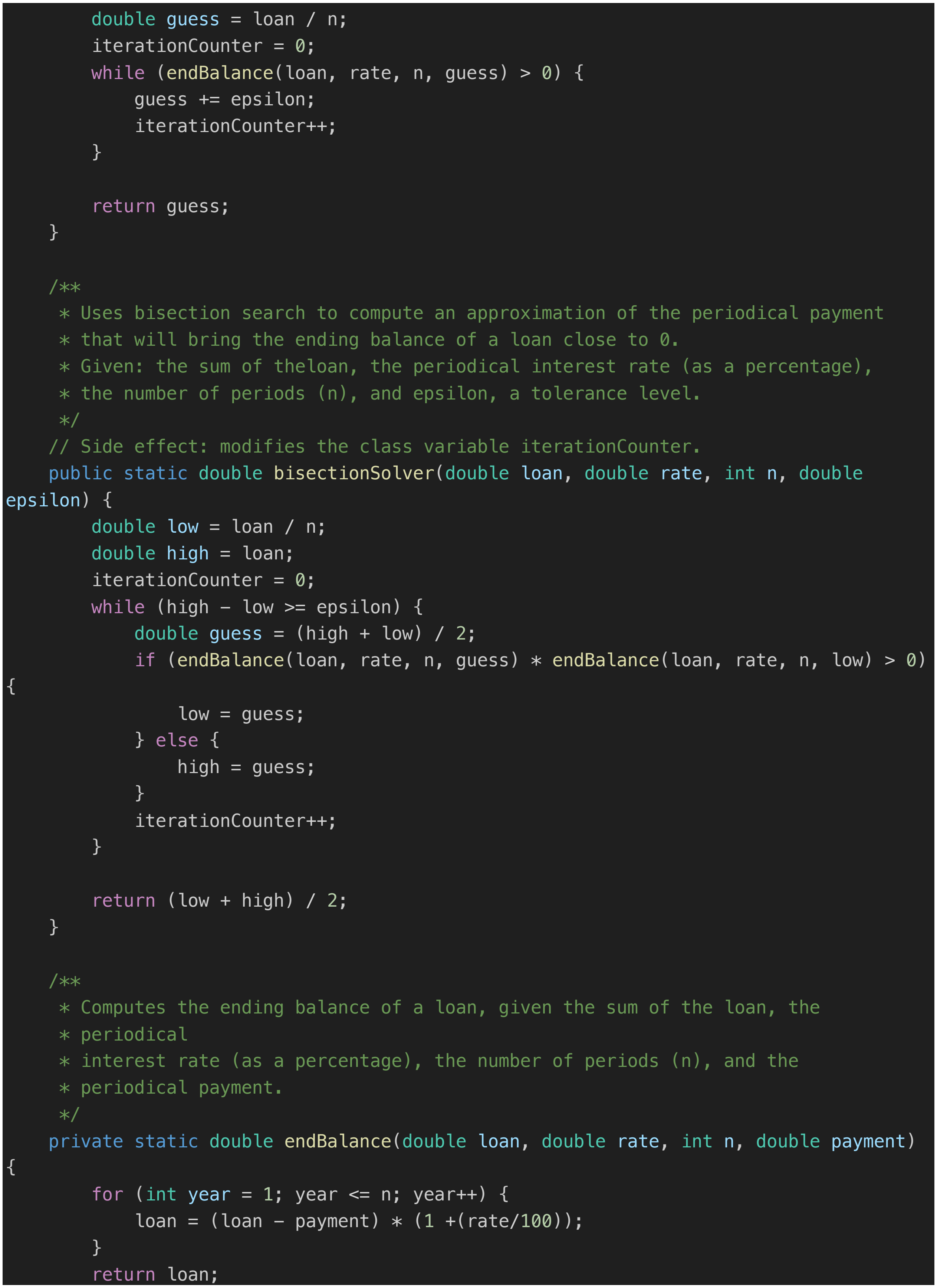
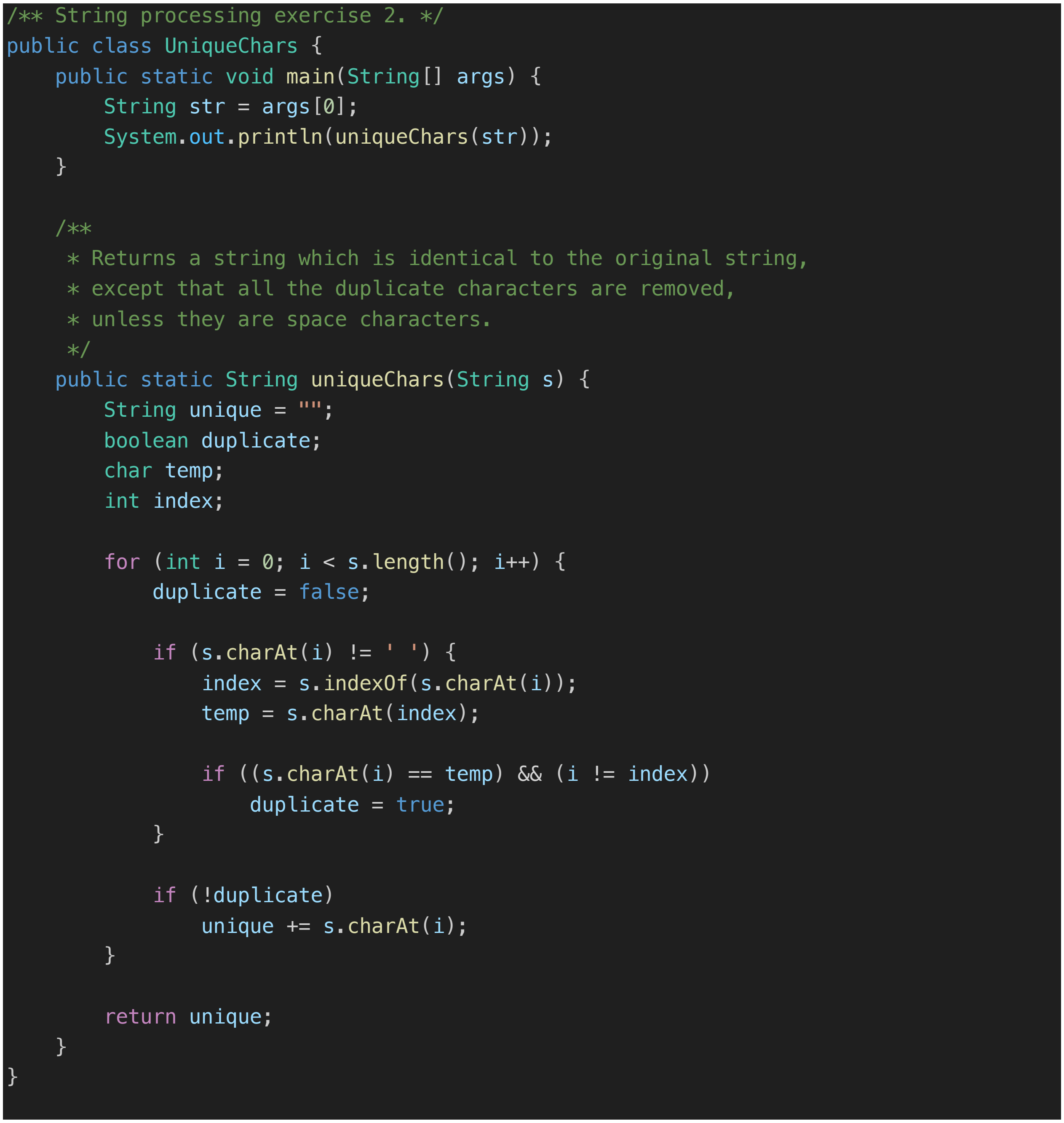
```
/**
 * Computes the periodical payment necessary to re-pay a given loan.
 */
public class LoanCalc {

    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) {
```

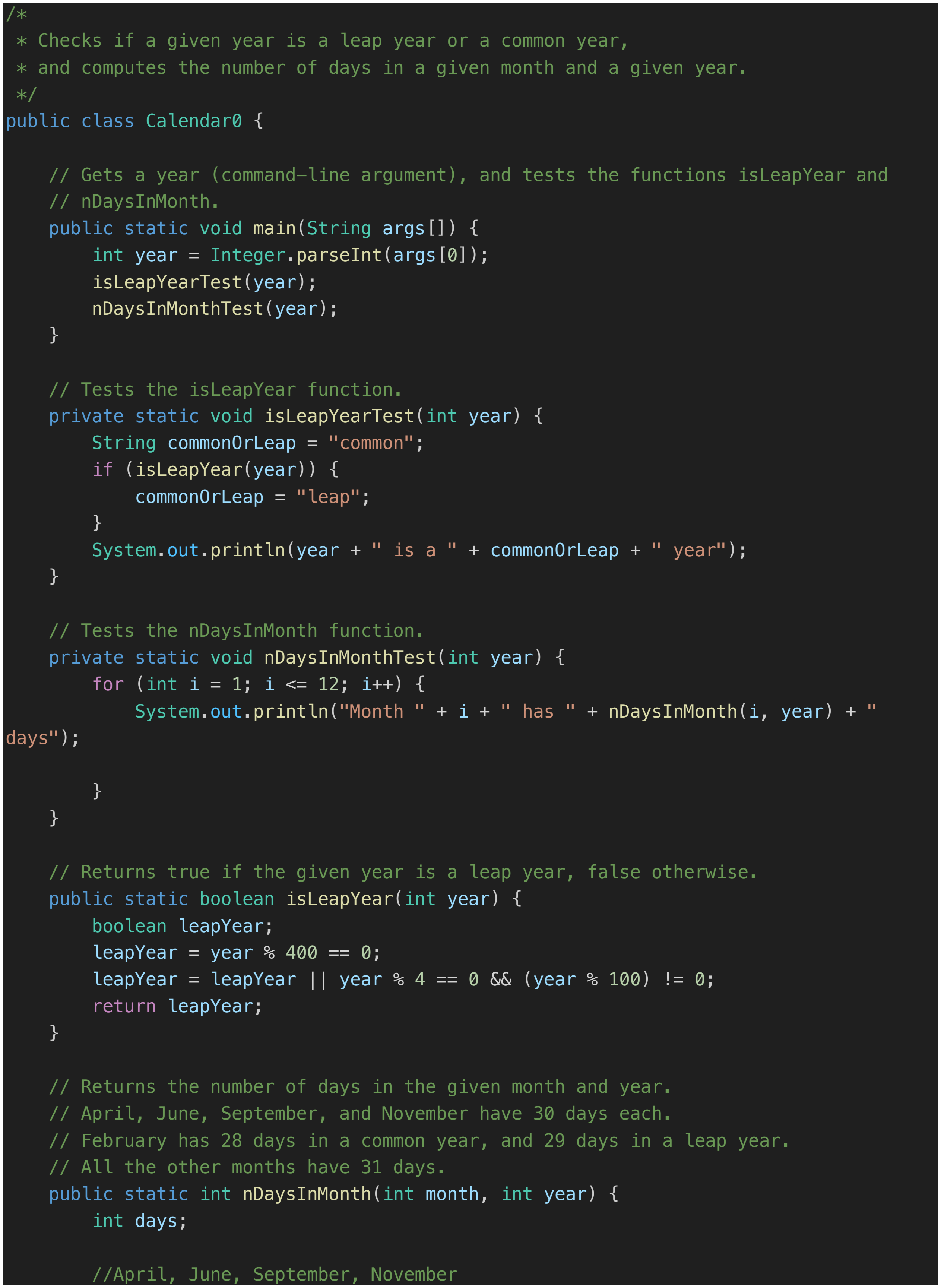


```
        double guess = loan / n;
        iterationCounter = 0;
        while (endBalance(loan, rate, n, guess) > 0) {
            guess += epsilon;
            iterationCounter++;
        }

        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) {
        double low = loan / n;
        double high = loan;
        iterationCounter = 0;
        while (high - low >= epsilon) {
            double guess = (high + low) / 2;
            if (endBalance(loan, rate, n, guess) * endBalance(loan, rate, n, low) > 0)
{
                low = guess;
            } else {
                high = guess;
            }
            iterationCounter++;
        }

        return (low + high) / 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)
{
        for (int year = 1; year <= n; year++) {
            loan = (loan - payment) * (1 +(rate/100));
        }
        return loan;
    }
}
```

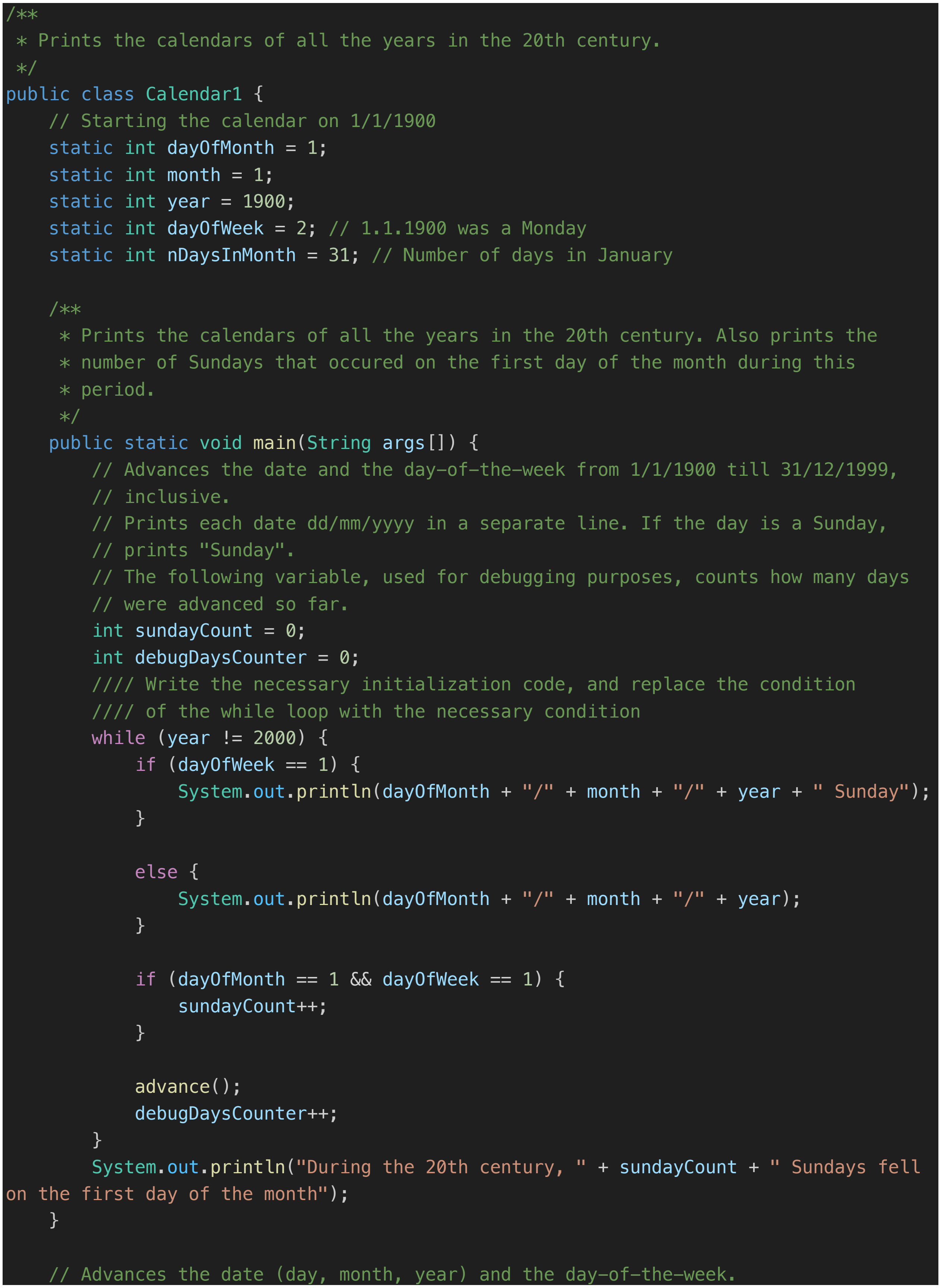
```
/** String processing exercise 1. */
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));

    }

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



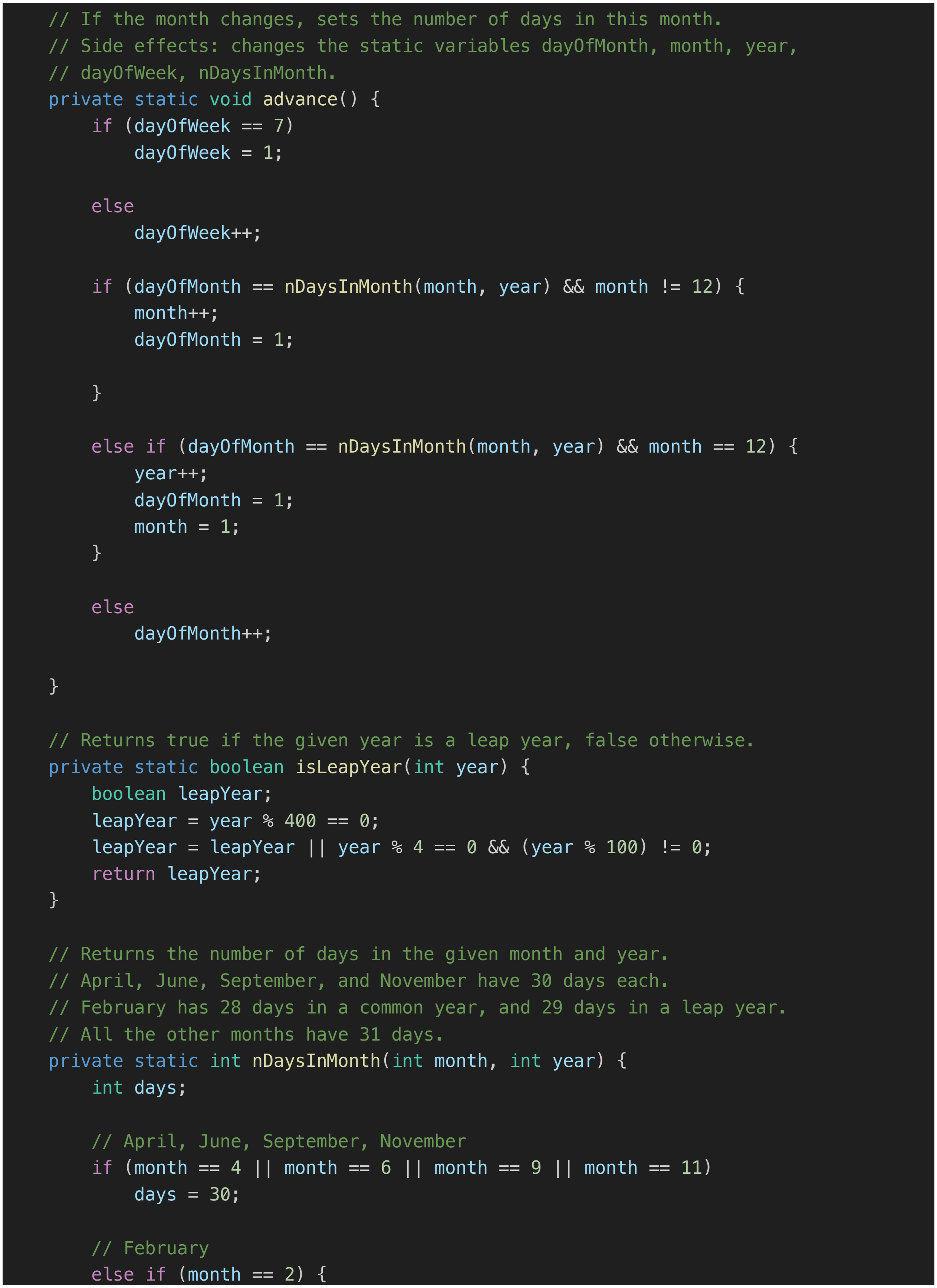
```java
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    /**
     * 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 unique = "";
        boolean duplicate;
        char temp;
        int index;

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

            if (s.charAt(i) != ' ') {
                index = s.indexOf(s.charAt(i));
                temp = s.charAt(index);

                if ((s.charAt(i) == temp) && (i != index))
                    duplicate = true;
            }

            if (!duplicate)
                unique += s.charAt(i);
        }

        return unique;
    }
}
```



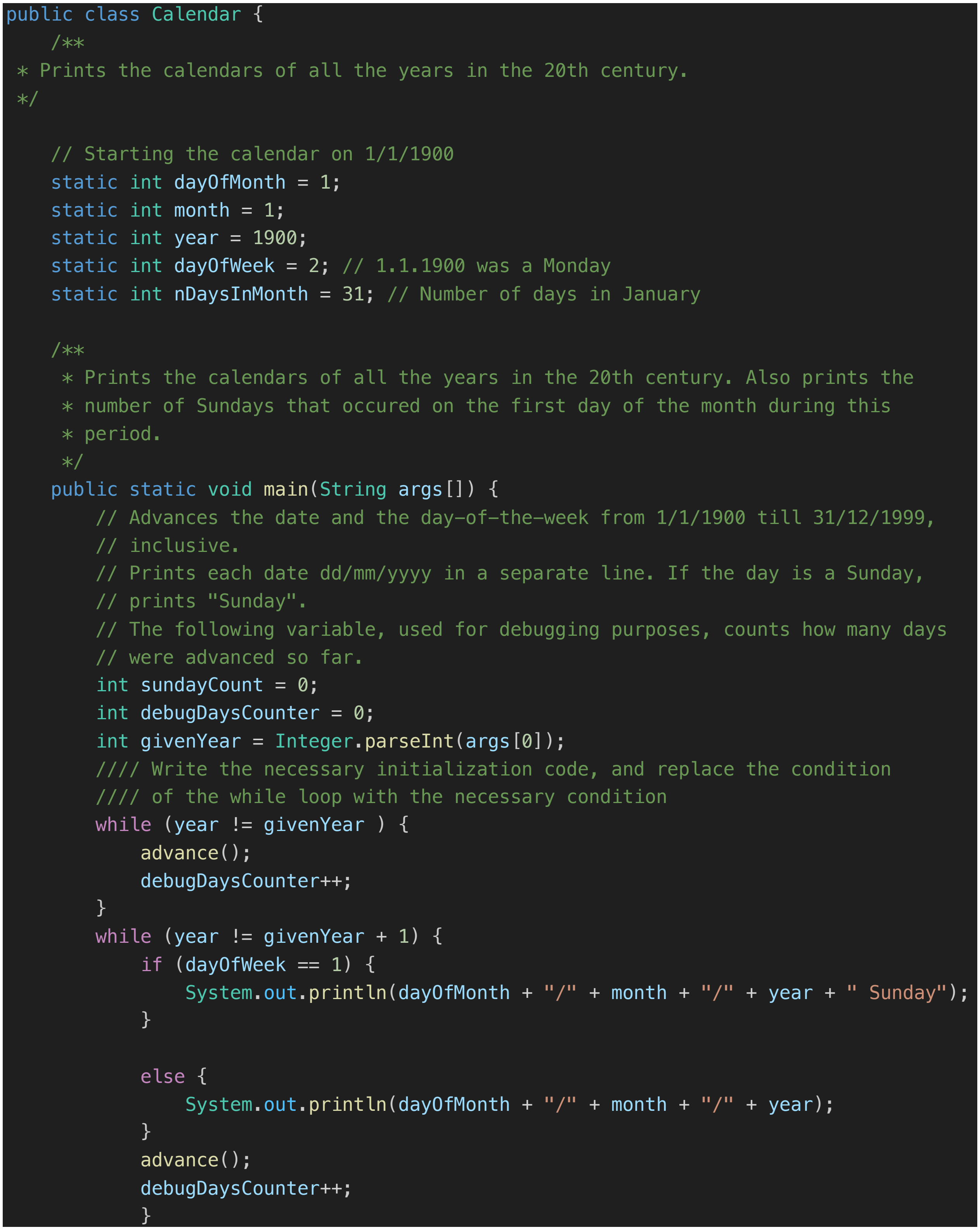
```
/*
 * Checks if a given year is a leap year or a common year,
 * and computes the number of days in a given month and a given year.
 */
public class Calendar0 {

    // 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) {
        boolean leapYear;
        leapYear = year % 400 == 0;
        leapYear = leapYear || year % 4 == 0 && (year % 100) != 0;
        return leapYear;
    }

    // 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) {
        int days;

        //April, June, September, November
        if( month == 4 || month == 6 || month == 9 || month == 11)
            days = 30;

            // February
        else if (month == 2) {

            if(isLeapYear(year))
                days = 29;
            else
                days = 28;
        }

        // January, March, May, July, August, October, December
        else
            days = 31;

        return days;
    }
}
```



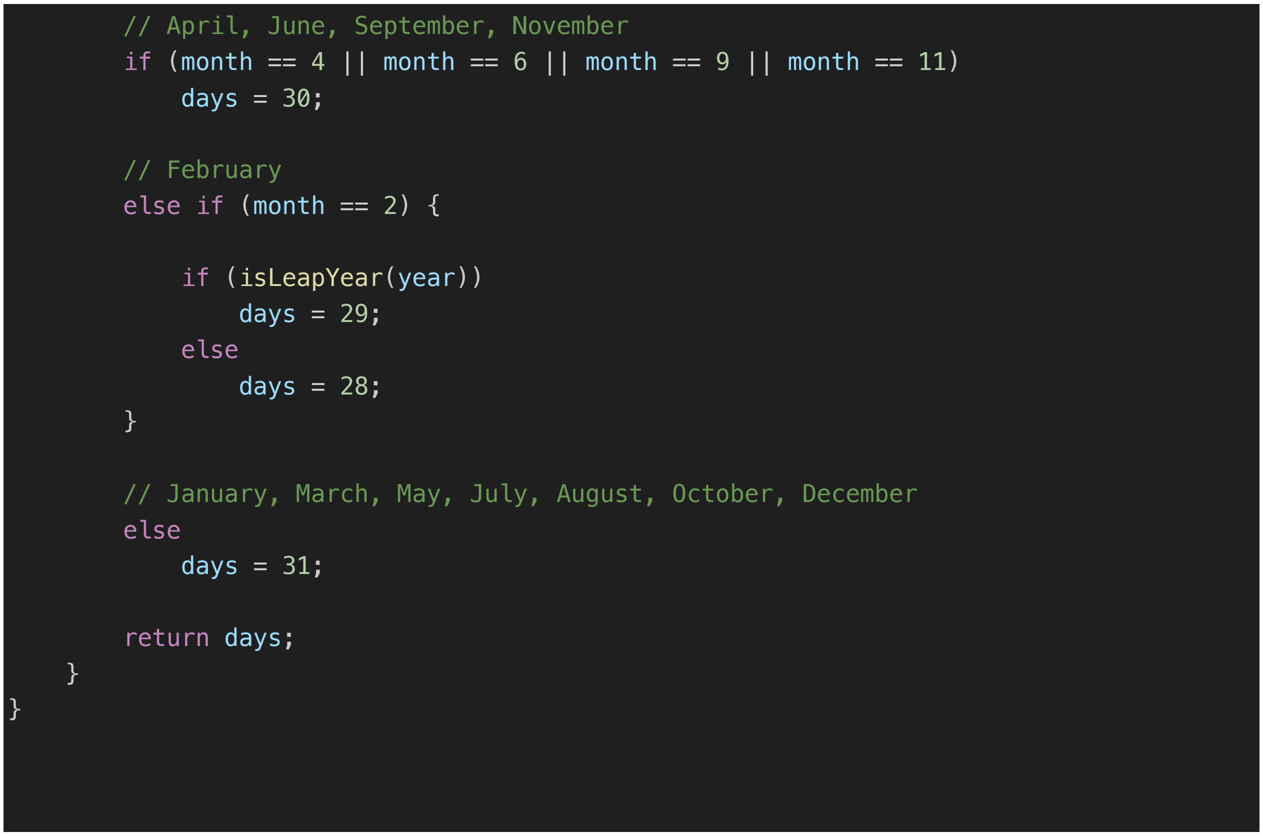
```
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar1 {
    // Starting the calendar on 1/1/1900
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2; // 1.1.1900 was a Monday
    static int nDaysInMonth = 31; // Number of days in January

    /**
     * Prints the calendars of all the years in the 20th century. Also prints the
     * number of Sundays that 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 sundayCount = 0;
        int debugDaysCounter = 0;
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year != 2000) {
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
            }

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

            if (dayOfMonth == 1 && dayOfWeek == 1) {
                sundayCount++;
            }

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

    // 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() {
        if (dayOfWeek == 7)
            dayOfWeek = 1;

        else
            dayOfWeek++;

        if (dayOfMonth == nDaysInMonth(month, year) && month != 12) {
            month++;
            dayOfMonth = 1;

        }

        else if (dayOfMonth == nDaysInMonth(month, year) && month == 12) {
            year++;
            dayOfMonth = 1;
            month = 1;
        }

        else
            dayOfMonth++;

    }

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

    // 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 days;

        // April, June, September, November
        if (month == 4 || month == 6 || month == 9 || month == 11)
            days = 30;

        // February
        else if (month == 2) {
            if (isLeapYear(year))
                days = 29;
            else
                days = 28;
        }

        // January, March, May, July, August, October, December
        else
            days = 31;

        return days;
    }
}
```



```java
public class Calendar {
    /**
 * Prints the calendars of all the years in the 20th century.
 */

    // 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 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 sundayCount = 0;
        int debugDaysCounter = 0;
        int givenYear = Integer.parseInt(args[0]);
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year != givenYear ) {
            advance();
            debugDaysCounter++;
        }
        while (year != givenYear + 1) {
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
            }

            else {
                System.out.println(dayOfMonth + "/" + month + "/" + year);
            }
            advance();
            debugDaysCounter++;
            }
    }



// 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() {
    if (dayOfWeek == 7)
        dayOfWeek = 1;

    else
        dayOfWeek++;

    if (dayOfMonth == nDaysInMonth(month, year) && month != 12) {
        month++;
        dayOfMonth = 1;

    }

    else if (dayOfMonth == nDaysInMonth(month, year) && month == 12) {
        year++;
        dayOfMonth = 1;
        month = 1;
    }

    else
        dayOfMonth++;

}

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

// 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 days;
```



```
        // April, June, September, November
        if (month == 4 || month == 6 || month == 9 || month == 11)
            days = 30;

        // February
        else if (month == 2) {

            if (isLeapYear(year))
                days = 29;
            else
                days = 28;
        }

        // January, March, May, July, August, October, December
        else
            days = 31;

        return days;
    }
}
```