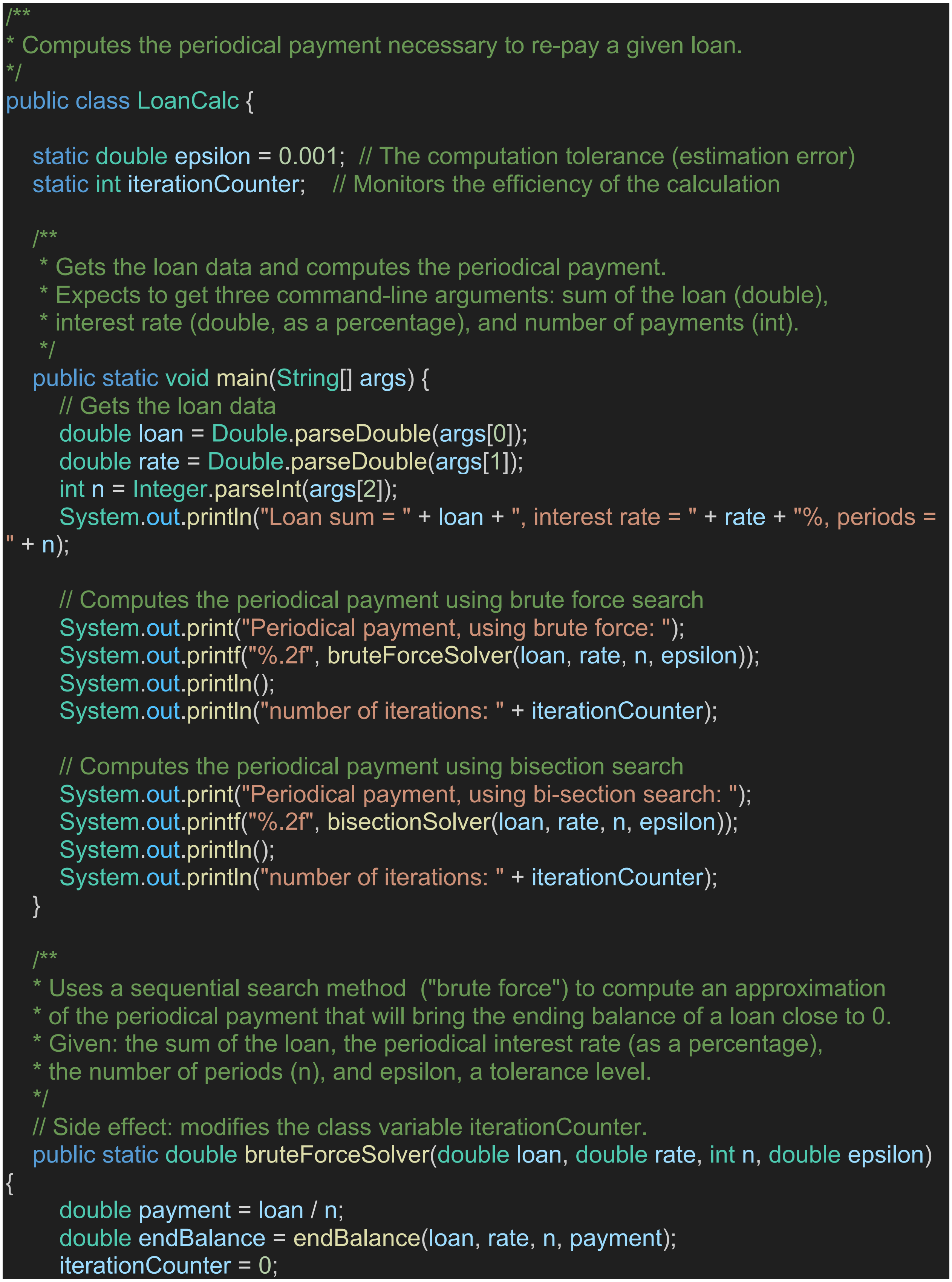
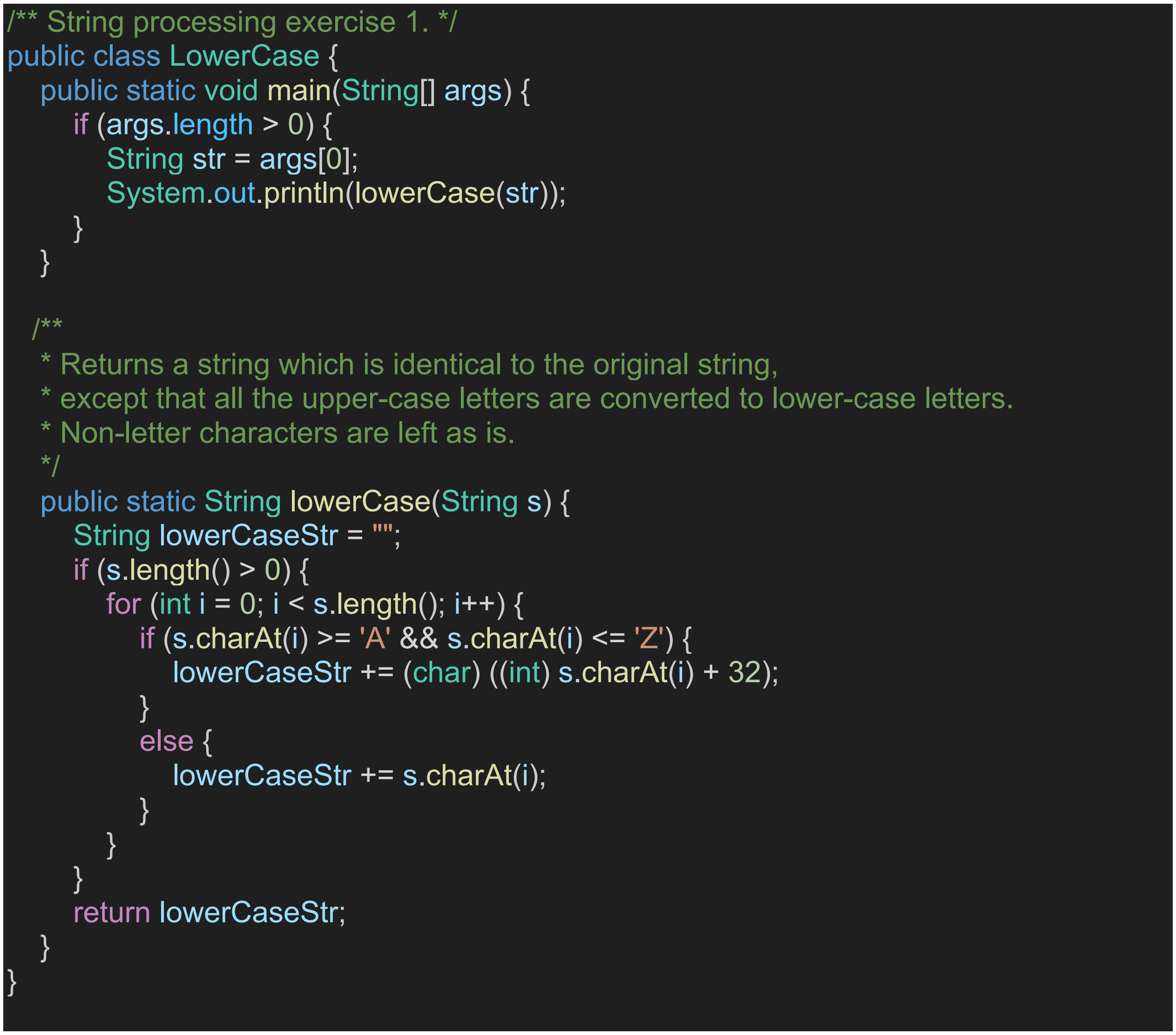
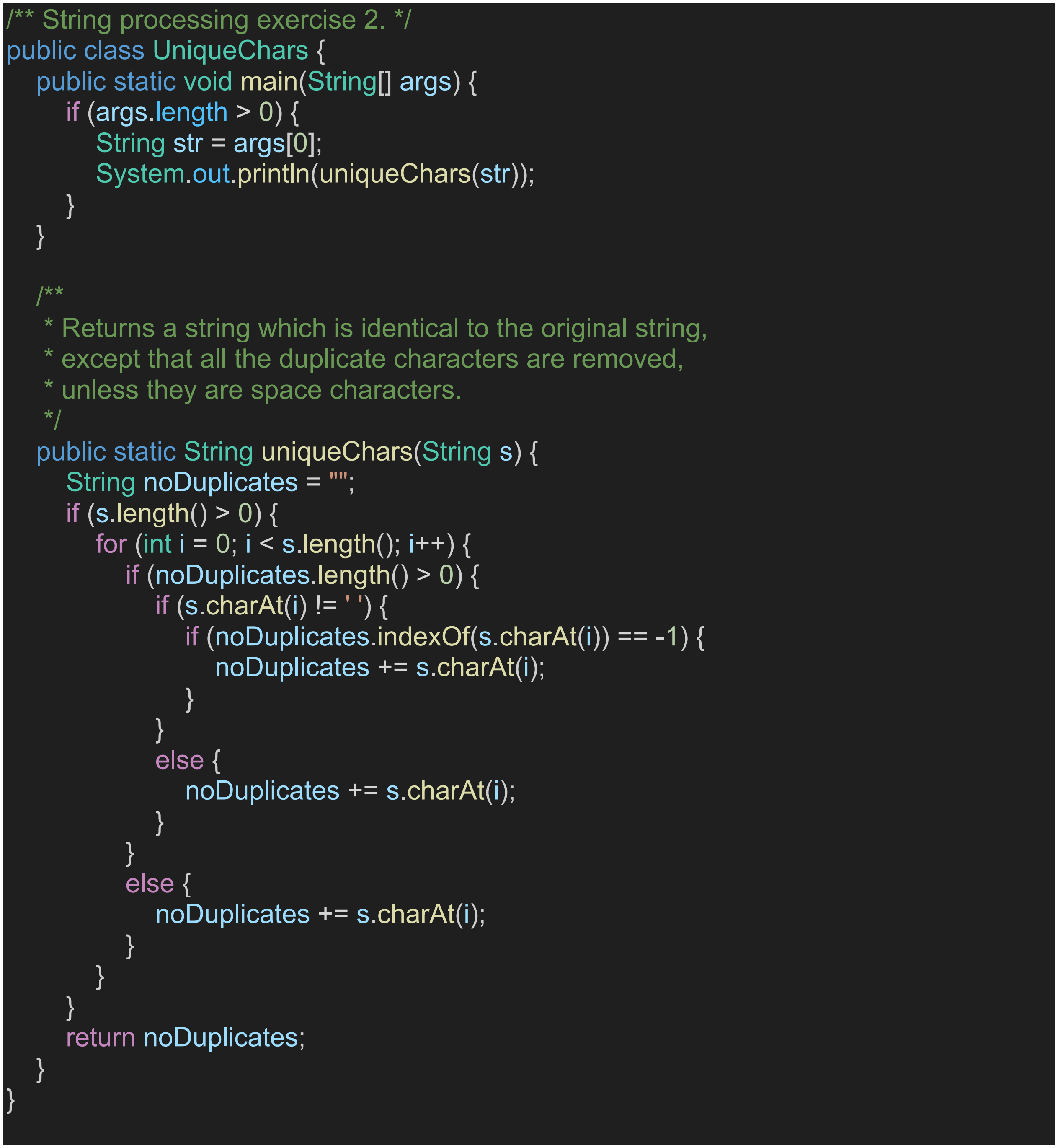


```
/**
* Computes the periodical payment necessary to re-pay a given loan.
*/
public class LoanCalc {

    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 payment = loan / n;
        double endBalance = endBalance(loan, rate, n, payment);
        iterationCounter = 0;
        while (endBalance - epsilon > 0) {
            payment += epsilon;
            endBalance = endBalance(loan, rate, n, payment);
            iterationCounter++;
        }
        return payment;
    }

    /**
    * 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 h = loan;
        double l = loan / n;
        double g = (h + l) / 2.0;
        double endLow = 0;
        double endHigh = 0;
        iterationCounter = 0;

        while (h - l > epsilon) {
            // h = g;
            // g = (h + l) / 2.0;
            endLow = endBalance(loan, rate, n, l);
            endHigh = endBalance(loan, rate, n, g);
            if (endHigh * endLow > 0) {
                l = g;
            }
            else {
                h = g;
            }
            g = (h + l) / 2.0;
            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 endBalanceOfLoan = loan;
        for (int i = 0; i < n; i++) {
            endBalanceOfLoan = (endBalanceOfLoan - payment) * (1.0 + (rate / 100.0));
            }
        return endBalanceOfLoan;
    }
}
```

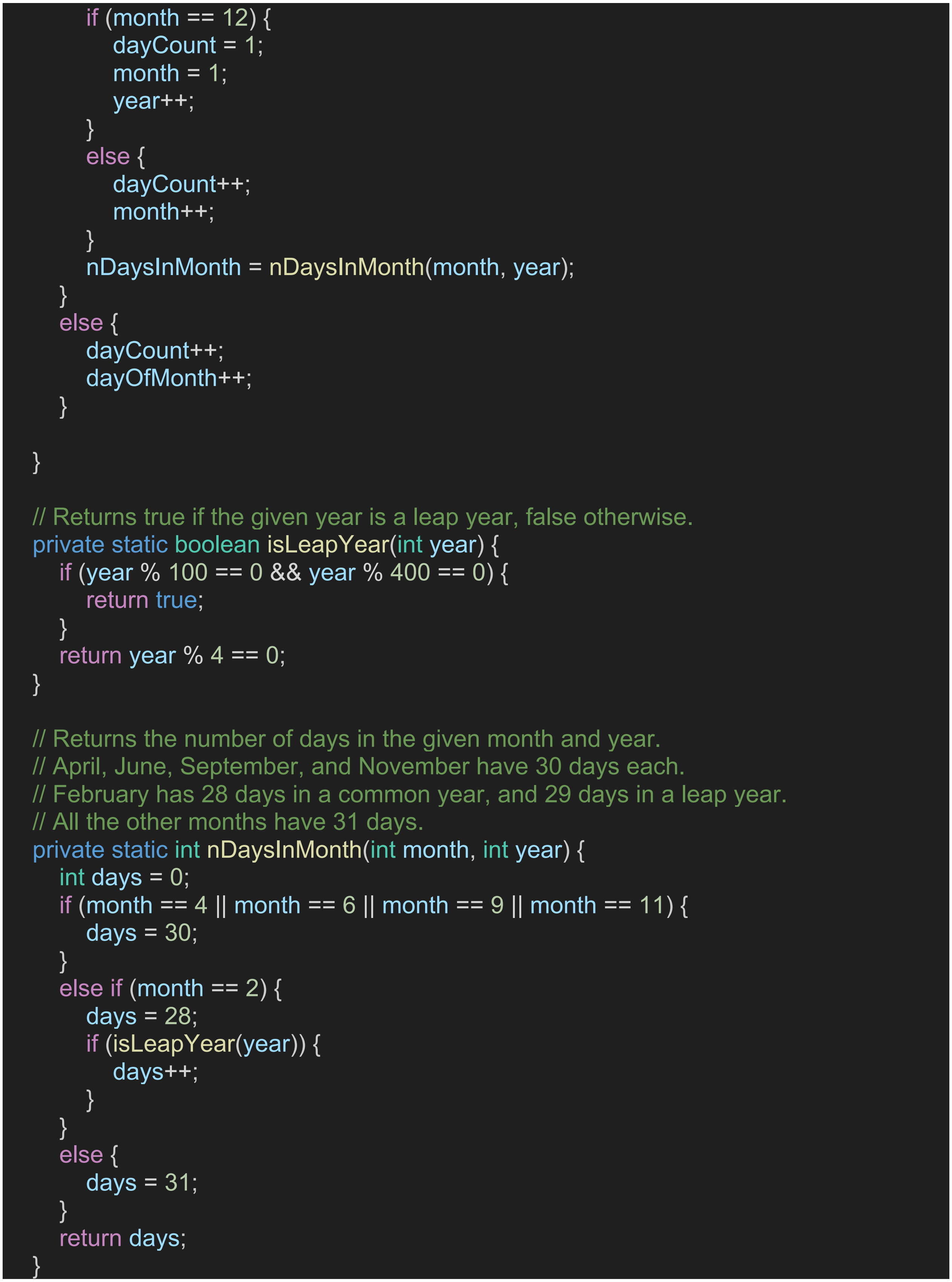


```java
/** String processing exercise 1. */
public class LowerCase {
    public static void main(String[] args) {
        if (args.length > 0) {
            String str = args[0];
            System.out.println(lowerCase(str));
        }
    }

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



```java
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        if (args.length > 0) {
            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 noDuplicates = "";
        if (s.length() > 0) {
            for (int i = 0; i < s.length(); i++) {
                if (noDuplicates.length() > 0) {
                    if (s.charAt(i) != ' ') {
                        if (noDuplicates.indexOf(s.charAt(i)) == -1) {
                            noDuplicates += s.charAt(i);
                        }
                    }
                    else {
                        noDuplicates += s.charAt(i);
                    }
                }
                else {
                    noDuplicates += s.charAt(i);
                }
            }
        }
        return noDuplicates;
    }
}
```

```
/**
 * Prints the calendars of all the years of a given year
 */
public class Calendar {
    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 dayCount = 1;

    public static void main(String args[]) {
        if (args.length > 0) {


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

            while ( year != yearToPrint ) {

                advance();
            }

            while ( year != yearToPrint + 1 ) {
                System.out.print(dayOfMonth + "/" + month + "/" + year);
                if (dayOfWeek == 1) {
                    System.out.print(" Sunday");
                }
                System.out.println();
                advance();
            }
        }
    }

    // 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) {
            dayOfMonth = 1;
```



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

}

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

// 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 = 0;
    if (month == 4 || month == 6 || month == 9 || month == 11) {
        days = 30;
    }
    else if (month == 2) {
        days = 28;
        if (isLeapYear(year)) {
            days++;
        }
    }
    else {
        days = 31;
    }
    return days;
}
```

```
}
```