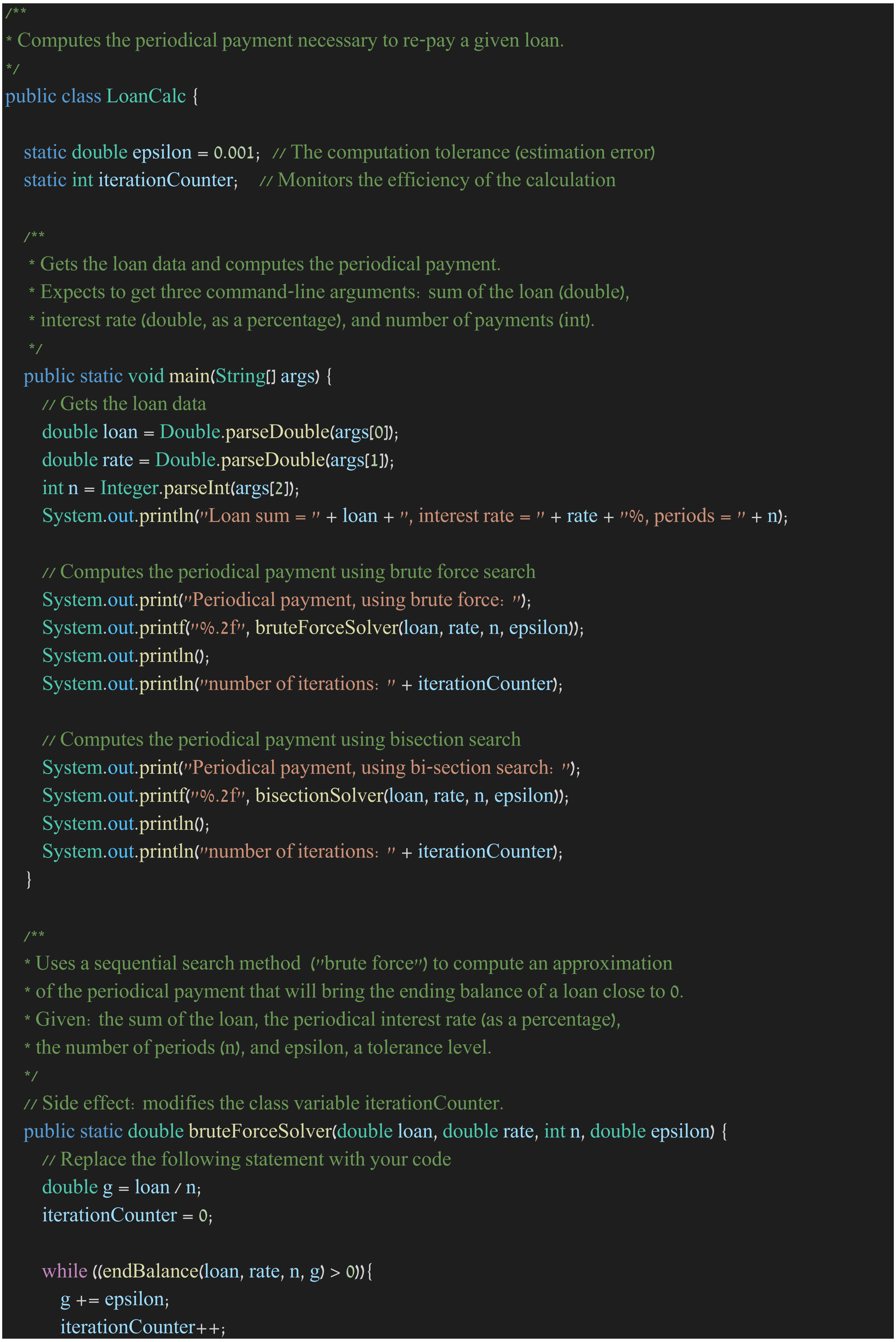
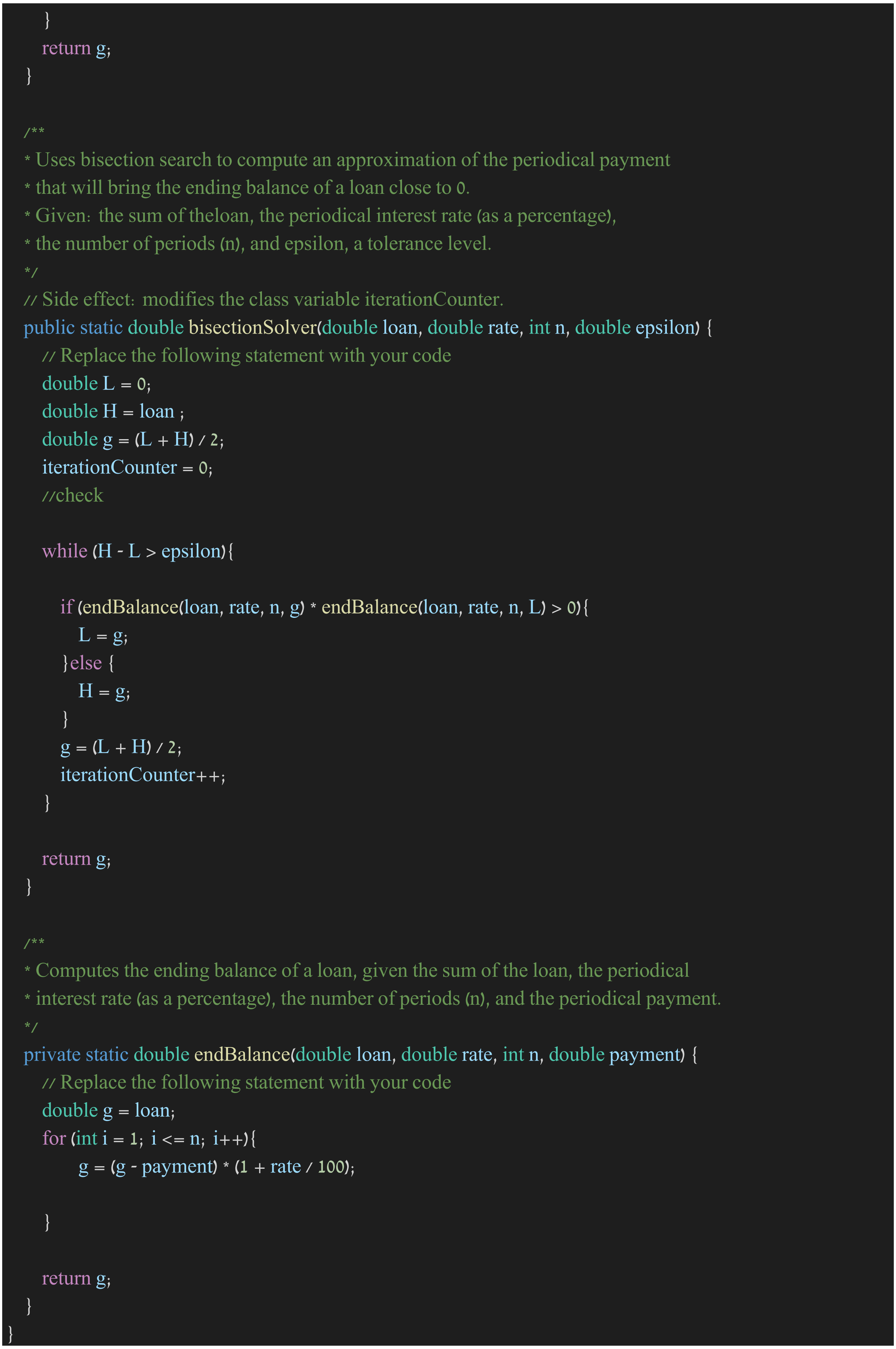
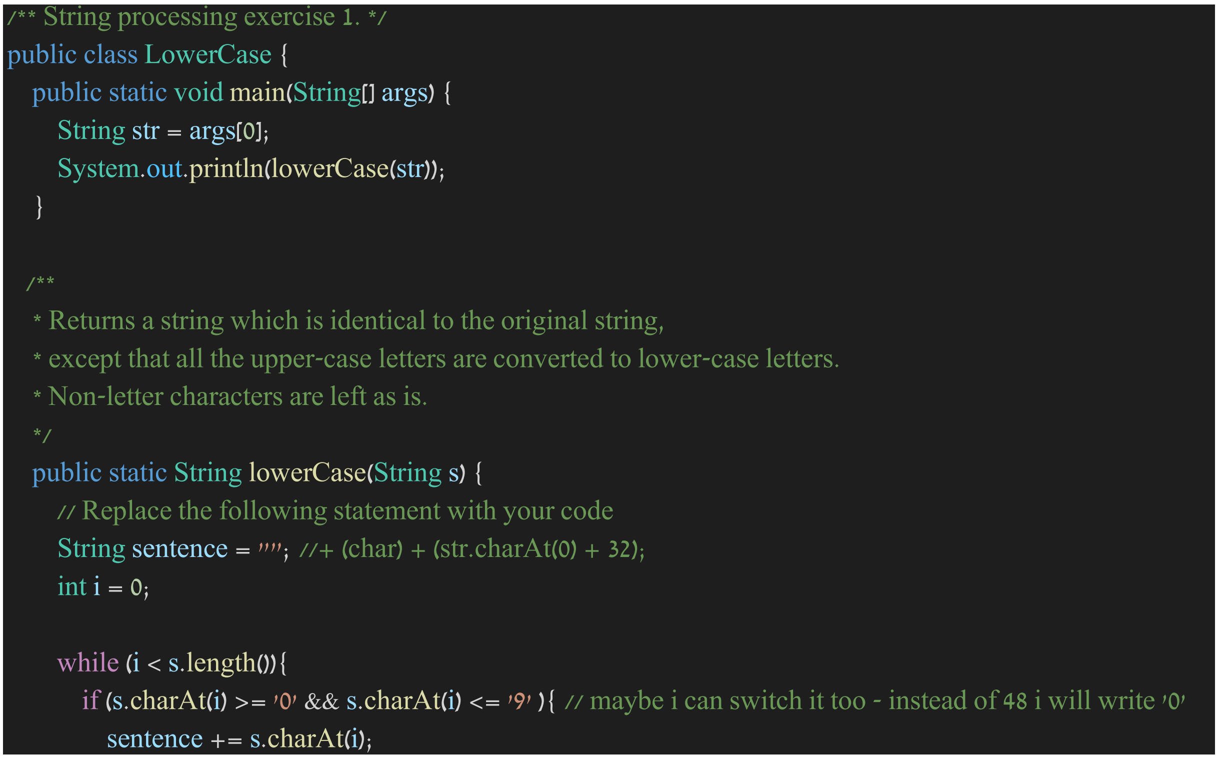


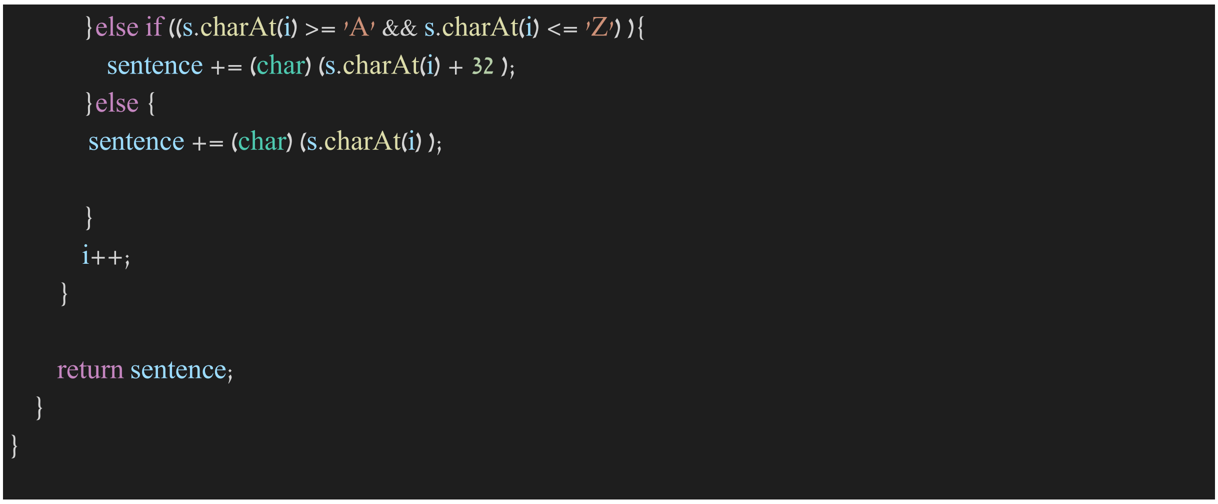
```
/**
* Computes the periodical payment necessary to re-pay a given loan.
*/
public class LoanCalc {

    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) > 0)){
            g += epsilon;
            iterationCounter++;
```



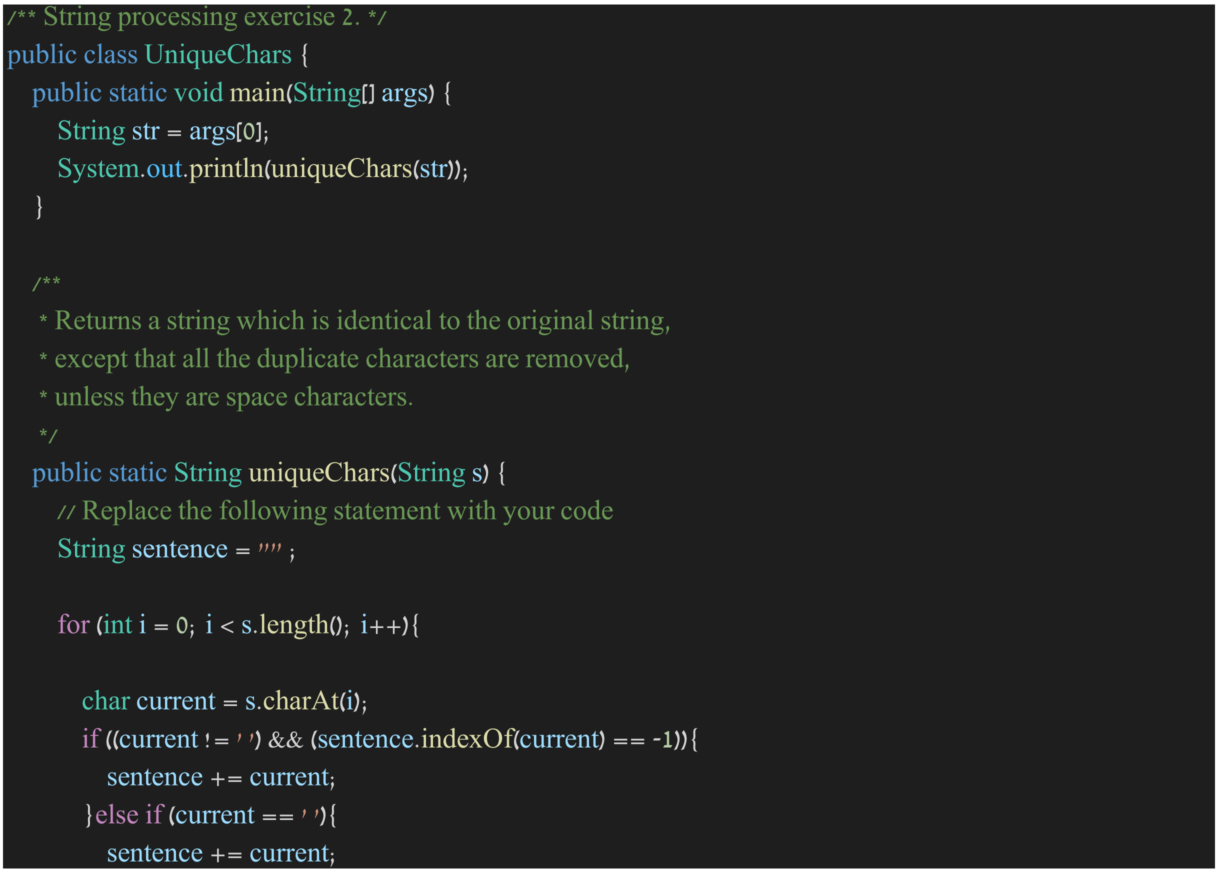
```
        }
        return g;
    }

    /**
    * 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) {
        // Replace the following statement with your code
        double L = 0;
        double H = loan ;
        double g = (L + H) / 2;
        iterationCounter = 0;
        //check

        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) {
        // Replace the following statement with your code
        double g = loan;
        for (int i = 1; i <= n; i++){
                g = (g - payment) * (1 + rate / 100);

        }

        return g;
    }
}
```



```java
/** 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
      String sentence = ""; //+ (char) + (str.charAt(0) + 32);
      int i = 0;

      while (i < s.length()){
         if (s.charAt(i) >= '0' && s.charAt(i) <= '9' ){ // maybe i can switch it too - instead of 48 i will write '0'
            sentence += s.charAt(i);
```

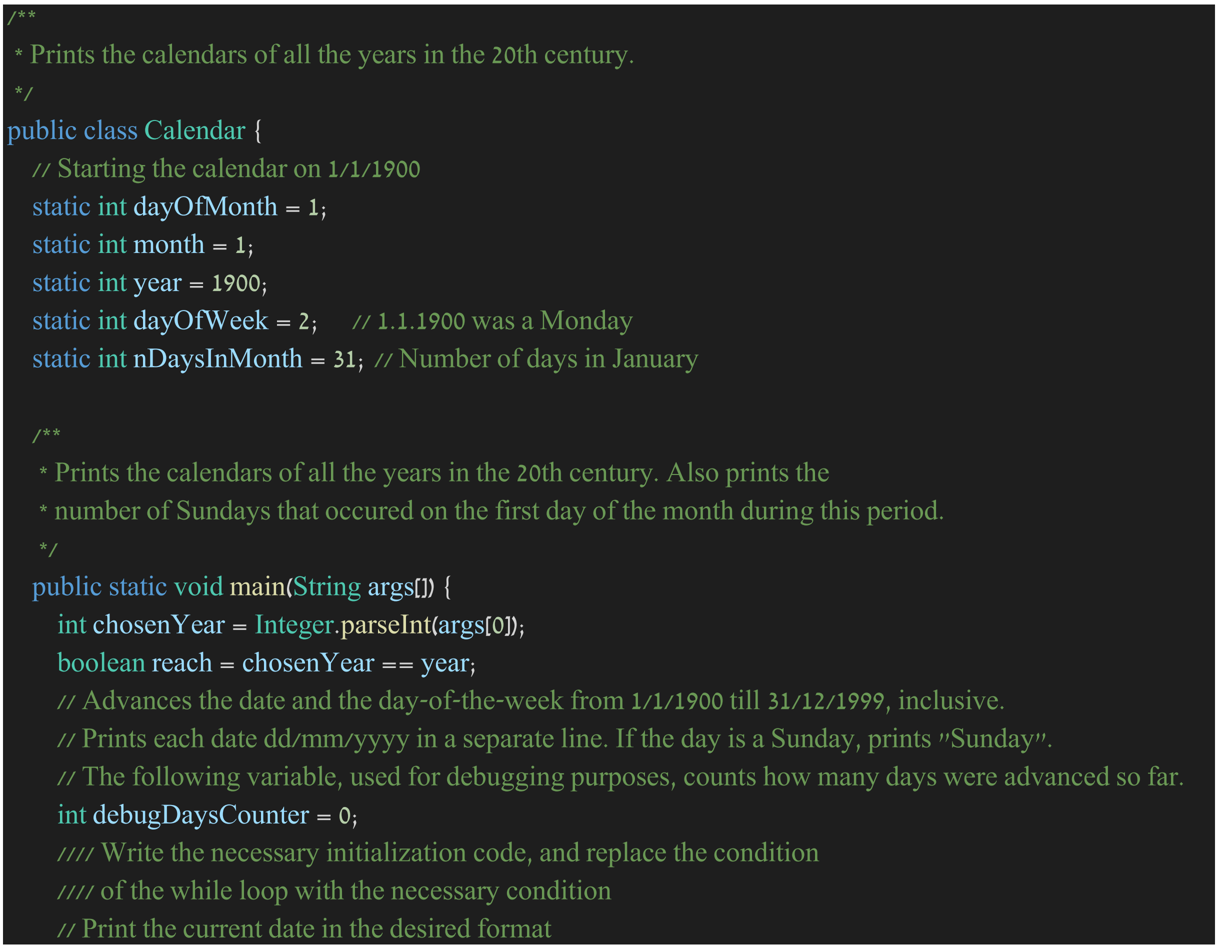


```
            }else if ((s.charAt(i) >= 'A' && s.charAt(i) <= 'Z') ){
                sentence += (char) (s.charAt(i) + 32 );
            }else {
            sentence += (char) (s.charAt(i) );

            }
            i++;
        }

        return sentence;
    }
}
```

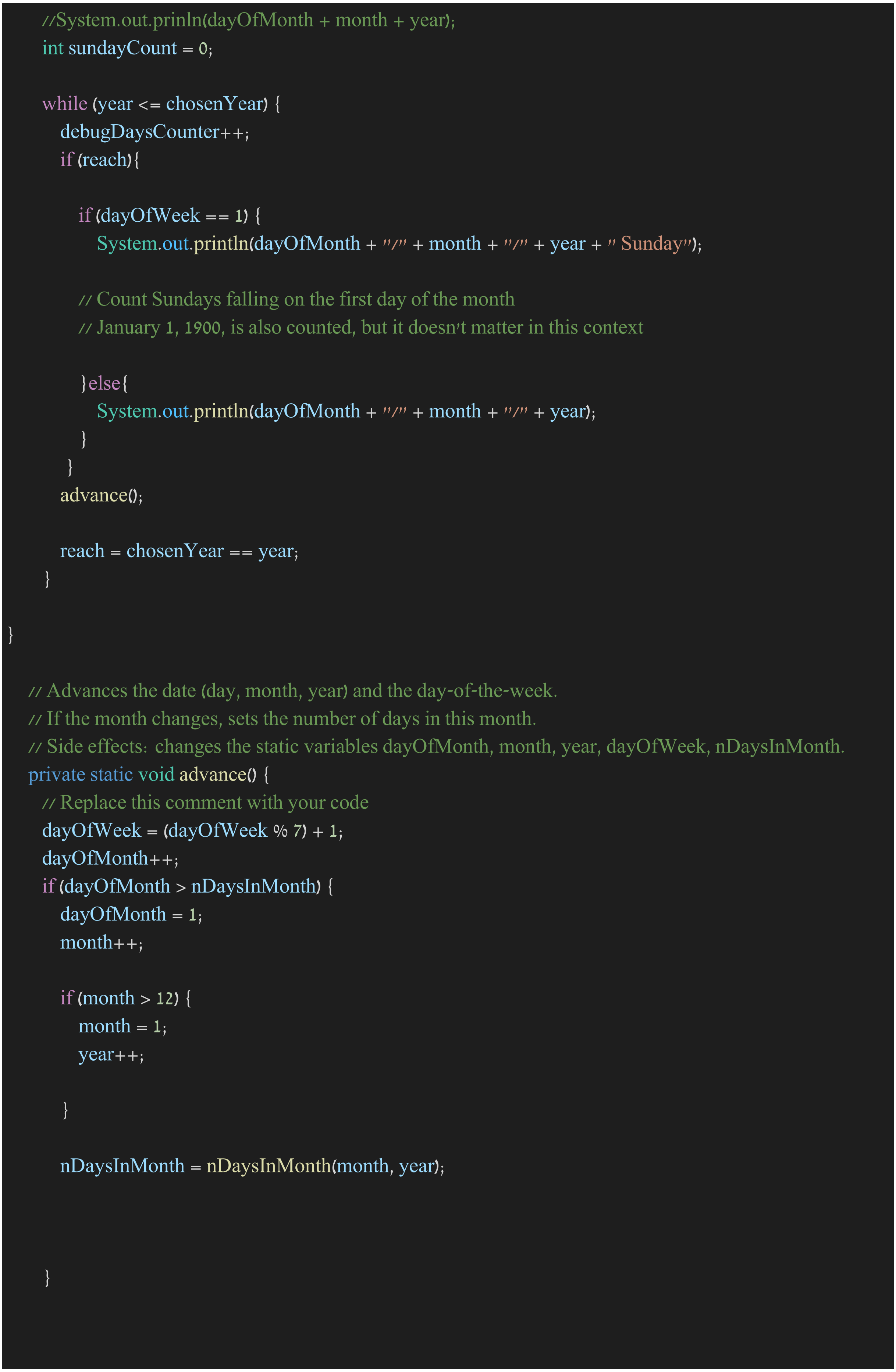


```
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    /**
     * 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 sentence = "" ;

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

            char current = s.charAt(i);
            if ((current != ' ') && (sentence.indexOf(current) == -1)){
                sentence += current;
            }else if (current == ' '){
                sentence += current;
```

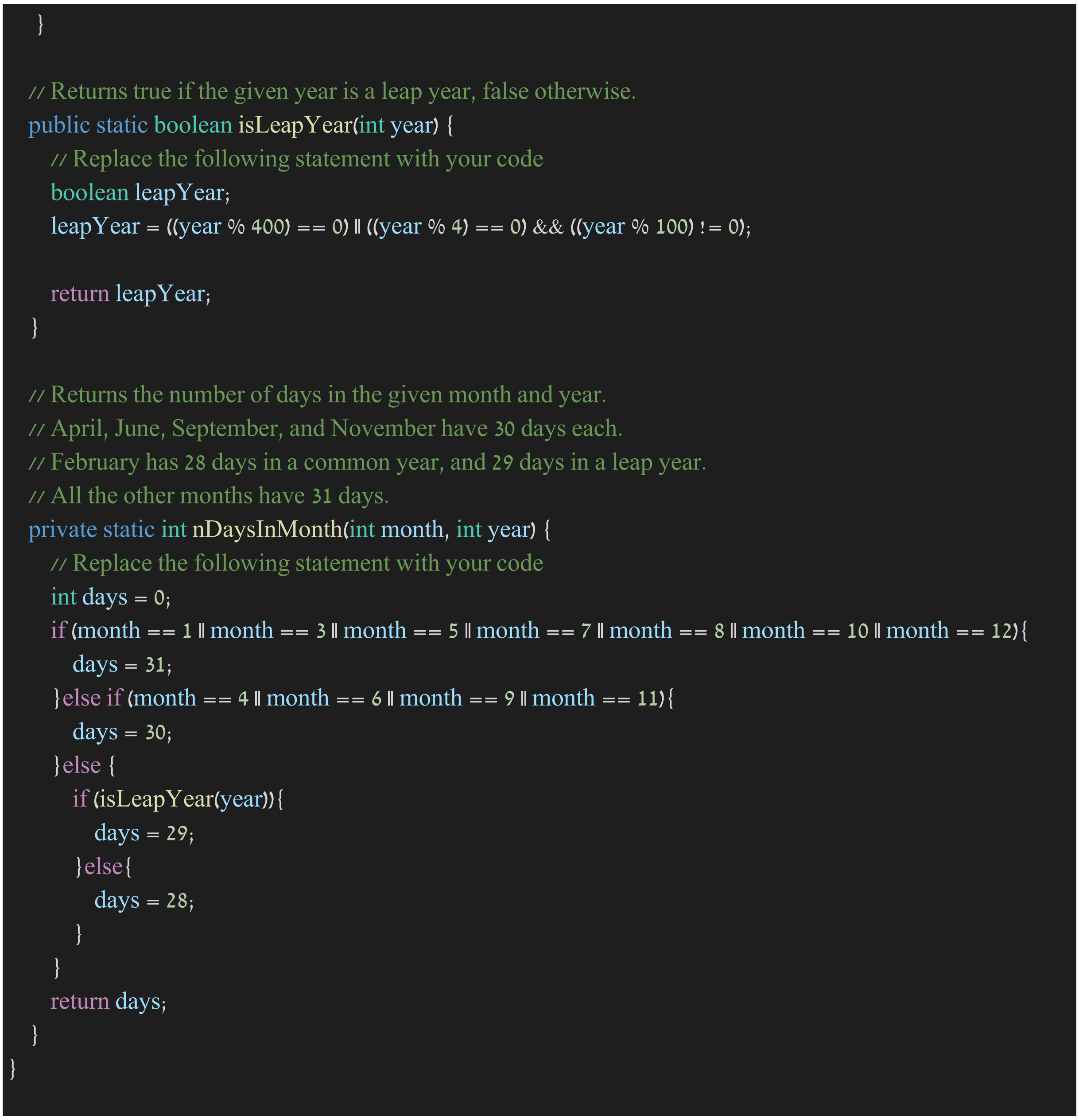
```
            }
        }

        return sentence;
    }
}
```



```
/**
 * 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 occured on the first day of the month during this period.
     */
    public static void main(String args[]) {
        int chosenYear = Integer.parseInt(args[0]);
        boolean reach = chosenYear == year;
        // 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 debugDaysCounter = 0;
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        // Print the current date in the desired format
```



```
    //System.out.prinln(dayOfMonth + month + year);
    int sundayCount = 0;

    while (year <= chosenYear) {
      debugDaysCounter++;
      if (reach){

        if (dayOfWeek == 1) {
          System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");

        // Count Sundays falling on the first day of the month
        // January 1, 1900, is also counted, but it doesn't matter in this context

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

      reach = chosenYear == year;
    }

}

  // 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
    dayOfWeek = (dayOfWeek % 7) + 1;
    dayOfMonth++;
    if (dayOfMonth > nDaysInMonth) {
      dayOfMonth = 1;
      month++;

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

      }

      nDaysInMonth = nDaysInMonth(month, year);



    }
```



```java
    }

    // 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 leapYear;
        leapYear = ((year % 400) == 0) || ((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) {
        // Replace the following statement with your code
        int days = 0;
        if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12){
            days = 31;
        }else if (month == 4 || month == 6 || month == 9 || month == 11){
            days = 30;
        }else {
            if (isLeapYear(year)){
                days = 29;
            }else{
                days = 28;
            }
        }
        return days;
    }
}
```