```
/**
* 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 paymentForAYear = (loan / n);
      double amountLeft = loan;
      while(amountLeft > epsilon) {
        paymentForAYear = paymentForAYear + epsilon;
        amountLeft = endBalance(loan, rate, n, paymentForAYear);
        iterationCounter ++;
      }
      return paymentForAYear;
    }

    /**
    * 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;
      iterationCounter = 0;
      while (H - L >= epsilon) {
        double paymentForAYear = (H + L) / 2;
        double amountLeft = endBalance(loan, rate, n, paymentForAYear) * endBalance(loan,
rate, n, L);
        if (amountLeft > 0) {
          L = paymentForAYear;
        } else {
          H = paymentForAYear;
        }
        iterationCounter++;
      }

      return (L + H) / 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) {
      double amountLeft = loan;
      for (int i = 0; i < n; i++) {
        amountLeft = (amountLeft - payment) * (1 + (rate / 100));
      }
      return amountLeft;
    }
}
```

```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) {
        String newString = "";
        for(int i = 0 ; i<s.length();i++){
            if(s.charAt(i) <=90 && s.charAt(i) >= 65){
                newString += (char) (s.charAt(i)+32);
            }
            else{
                newString+= s.charAt(i);
            }
        }
        // Replace the following statement with your code
        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 newString = "";
        for(int i = 0 ; i<s.length(); i++){
            if(newString.indexOf(s.charAt(i)) == -1 || s.charAt(i) == 32){
                newString += s.charAt(i);
            }
        }
        return newString;
    }
}
```

```java
/*
 * 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.printf("Month %d has %d days\n",i,nDaysInMonth(i,year));
    }
  }

  // Returns true if the given year is a leap year, false otherwise.
  public static boolean isLeapYear(int year) {
     return year %400 == 0 || ((year%4 == 0 ) && year%100 !=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.
  public static int nDaysInMonth(int month, int year) {
    switch (month){
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12:
        return 31;
      case 2:
        if(isLeapYear(year)){
          return 29;
        }
        else{
          return 28;
        }
      case 4:
      case 6:
      case 9:
      case 11:
        return 30;
    }
    return -1;
  }
}
```

```
/**
 * 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 debugDaysCounter = 0;
    int numberOfFirstOnSunday = 0;
     //// Write the necessary initialization code, and replace the condition
     //// of the while loop with the necessary condition
    while (year<2000) {
      nDaysInMonth = nDaysInMonth(month,year);
      if(dayOfMonth == 1){
        if(dayOfWeek == 1){
          numberOfFirstOnSunday++;
        }
        System.out.println(dayOfMonth + "/" + month + "/" + year
            + " Sunday");
      }
```

```
       else{
         System.out.println(dayOfMonth + "/" + month + "/" +
             year);
       }
      //// Write the body of the while
      advance();
      debugDaysCounter++;
      //// If you want to stop the loop after n days, replace the condition of the
      //// if statement with the condition (debugDaysCounter == n)
      if (false) {
        break;
      }
     }
    System.out.println("During the 20th century, " +
        numberOfFirstOnSunday + " Sundays fell on the first day of the month");
    //// Write the necessary ending code here
   }

   // 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 ++;
     } else {
       dayOfWeek = 1 ;
     }
     if (dayOfMonth < nDaysInMonth) {
       dayOfMonth++ ;
     }
     else{
       dayOfMonth=1 ;
       if(month<12){
```

```
      month++ ;
    }
    else{
      month = 1;
      year += 1;
    }
  }
}

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
  return year %400 == 0 || ((year%4 == 0 ) && year%100 !=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) {
  switch (month){
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      return 31;
    case 2:
      if(isLeapYear(year)){
        return 29;
      }
      else{
        return 28;
        }
      case 4:
      case 6:
      case 9:
      case 11:
        return 30;
    }
    return -1;
  }
}
```

```
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

    /**
     * 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 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) {
            nDaysInMonth = nDaysInMonth(month,year);
            if(year == givenYear){
                if(dayOfWeek == 1){
                    System.out.println(dayOfMonth + "/" + month + "/" + year
                            + " Sunday");
                }
                else{
                    System.out.println(dayOfMonth + "/" + month + "/" +
                            year);
                }
            }
            advance();
            //// If you want to stop the loop after n days, replace the condition of the
            //// if statement with the condition (debugDaysCounter == n)
            if (false) {
                break;
            }
        }
    }

    // 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 ++;
        } else {
            dayOfWeek = 1 ;
        }
        if (dayOfMonth < nDaysInMonth) {
            dayOfMonth++ ;
        }
        else{
            dayOfMonth=1 ;
            if(month<12){
                month++ ;
            }
            else{
                month = 1;
                year += 1;
            }
        }
    }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        return year %400 == 0 || ((year%4 == 0 ) && year%100 !=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) {
        switch (month){
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                return 31;
            case 2:
                if(isLeapYear(year)){
                    return 29;
                }
                else{
                    return 28;
                }
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
        }
        return -1;
    }
}
```