```
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);

     iterationCounter = 0;

     // 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;

    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
    iterationCounter = 0;
    double L = loan / n;
```

```
        double H = loan;

        double g = (L + H) / 2;

        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
        for (int i = 0; i < n; i++) {
            loan = (loan - payment) * (1 + rate / 100);
            //System.out.println("Increment " + (i + 1) + ": " + loan);
        }
        return loan;
    }
}
```

```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 str = "";

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

      if ((c >= 65) && (c <= 90)) {
        str += (char) (c + 32);
      }
      else {
        str += c;
      }
    }
    return str;
  }
}
```

```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 oldStr) {
    // Define "newStr" which we will add to to make the string without repeats
    String newStr = "";

    // Define boolean "repeatChar" which will become true if the program detects that there is a repeat character
    boolean repeatChar;

    for (int i = 0; i < oldStr.length(); i++) {
      // Initialize repeatChar to false at the start of each loop
      repeatChar = false;

      // Set currentChar to be the i'th character in the original string
      char currentChar = oldStr.charAt(i);

      //System.out.println(i + ": " + currentChar);

      // In this second loop we check for duplicates.
```

```
        // To make sure there are no duplicates, the nested loop will loop through the
new string to see if it already
        // contains that character. If it does, repeatChar is ticked true.
        for (int j = 0; j < newStr.length() && newStr.length() > 0; j++) {
            if (currentChar == newStr.charAt(j)) {
                repeatChar = true;
            }
        }

        // If repeatChar is ticked true, it does not add the i'th character to the new string.
        // If the current character is space, then it will add the i'th character anyway.
        // 32 is space in ASCI
        if ((repeatChar == false) || (currentChar == 32)) {
            newStr = newStr + currentChar;
        }
    }
    return newStr;
  }
}
```

```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) {
     int i = 1;
     while (i <= 12) {
        nDaysInMonth(i, year);
        i++;
     }
  }
```

```java
// Returns true if the given year is a leap year, false otherwise.
public static boolean isLeapYear(int year) {
    boolean leapYear = false;

    // Code derived from 1-2
    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 = 0;
    switch (month) {
        case 1:
            days = 31;
            break;
        case 2:
            days = 28;
            if (isLeapYear(year)) {
                days = 29;
            }
            break;
        case 3:
            days = 31;
            break;
```

```
        case 4:
            days = 30;
            break;
        case 5:
            days = 31;
            break;
        case 6:
            days = 30;
            break;
        case 7:
            days = 31;
            break;
        case 8:
            days = 31;
            break;
        case 9:
            days = 30;
            break;
        case 10:
            days = 31;
            break;
        case 11:
            days = 30;
            break;
        case 12:
            days = 31;
            break;
        default:
            days = 0;
            break;
    }
```

```
        System.out.println("Month " + month + " has " + days + " days");
        return 0;
    }
}
```

```java
/**
* 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;
     String isSunday = "";
     int firstSundayCounter = 0;
     //// Write the necessary initialization code, and replace the condition
     //// of the while loop with the necessary condition
     while (true) {
        isSunday = "";
        if (dayOfWeek == 1) {
           isSunday = " Sunday";
```

```
            if (dayOfMonth == 1) {
                firstSundayCounter++;
            }
        }
        System.out.println(dayOfMonth + "/" + month + "/" + year + isSunday);
        advance();
        debugDaysCounter++;
        //// If you want to stop the loop after n days, replace the condition of the
        if (debugDaysCounter == 36524) {
            System.out.println();
            System.out.println("During the 20th century, " + firstSundayCounter + "
Sundays fell on the first day of the month.");
            break;
        }
    }
    //// 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() {
    // Replace this comment with your code

    nDaysInMonth = nDaysInMonth(month, year);

    dayOfWeek++;
    dayOfWeek %= 7;
    dayOfMonth++;
```

```
    if (dayOfMonth > nDaysInMonth) {
        month++;
        dayOfMonth = 1;
    }

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

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    boolean leapYear = false;

    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 = 0;
    switch (month) {
        case 1:
            days = 31;
```

```
            break;
        case 2:
            days = 28;
            if (isLeapYear(year)) {
                days = 29;
            }
            break;
        case 3:
            days = 31;
            break;
        case 4:
            days = 30;
            break;
        case 5:
            days = 31;
            break;
        case 6:
            days = 30;
            break;
        case 7:
            days = 31;
            break;
        case 8:
            days = 31;
            break;
        case 9:
            days = 30;
            break;
        case 10:
            days = 31;
            break;
```

```
        case 11:
            days = 30;
            break;
        case 12:
            days = 31;
            break;
        default:
            days = 0;
            break;
    }
    return days;
  }
}
```

```java
/**
* 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[]) {
     // The following variable, used for debugging purposes, counts how many days
were advanced so far.
     int inputYear = Integer.parseInt(args[0]);
     String isSunday = "";

     while (true) {
        isSunday = "";
        if (dayOfWeek == 1) {
           isSunday = " Sunday";
        }

        if (year == inputYear) {
           System.out.println(dayOfMonth + "/" + month + "/" + year + isSunday);
        }
        advance();
```

```
        //// Stop looping when the year flips
        if (year > inputYear) {
            break;
        }
    }
    //// 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() {
    // Replace this comment with your code

    nDaysInMonth = nDaysInMonth(month, year);

    dayOfWeek++;
    dayOfWeek %= 7;

    dayOfMonth++;
    if (dayOfMonth > nDaysInMonth) {
        month++;
        dayOfMonth = 1;
    }

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

```
 }

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    boolean leapYear = false;

    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 = 0;
    switch (month) {
        case 1:
            days = 31;
            break;
        case 2:
            days = 28;
            if (isLeapYear(year)) {
                days = 29;
            }
            break;
        case 3:
            days = 31;
            break;
```

```
        case 4:
            days = 30;
            break;
        case 5:
            days = 31;
            break;
        case 6:
            days = 30;
            break;
        case 7:
            days = 31;
            break;
        case 8:
            days = 31;
            break;
        case 9:
            days = 30;
            break;
        case 10:
            days = 31;
            break;
        case 11:
            days = 30;
            break;
        case 12:
            days = 31;
            break;
        default:
            days = 0;
            break;
    }
```

```
    return days;
  }
}
```