```java
/**
* 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);
  }
```

```java
  /**
  * 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
      iterationCounter = 0;
      double g = loan/n;
      double guess = endBalance(loan, rate, n, g);
      while (guess > 0) {
          g += epsilon;
          guess = endBalance(loan, rate, n, g);
          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 H = loan;
      double L = 0.0;
```

```
        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
        double leftToPay = loan;
        for (int i = 0; i < n; i++) {
            leftToPay = (leftToPay -payment)*(1.0+rate/100);
        }
        return leftToPay;
    }
}
```

```java
public class LowerCase {
  public static void main(String[] args) {
    /*  if (args.length == 0) {
          System.out.println("No argument provided.");
          return; // Exit the program if no arguments are provided
      }*/

      String str = args[0];
      int length = str.length();
      int x = 0;

      while (x < length) {
          char L = str.charAt(x);
          String letter = Character.toString(L);

          if (Character.isUpperCase(L)) {
              letter = letter.toLowerCase();
          }

          System.out.print(letter);
          x++;
      }
  }
}
```

```java
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 str) {
     String fin = "";
     for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        boolean duplicate = false;
        for (int j = 0; j < fin.length(); j++) {
           if (fin.charAt(j) == ch) {
              duplicate = true;
              break;
           }
        }
        if (!duplicate || ch == ' ') {
           fin += ch;
        }
     }
     return fin;
  }
}
```

```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++) {
        int daysInMonth = nDaysInMonth(i,year);
        System.out.println("Month " + i + " has " + daysInMonth + " days");
     }
  }
  // Returns true if the given year is a leap year, false otherwise.
  public static boolean isLeapYear(int year) {
     if ((year%400 != 0) && (year%100 == 0)) {
        return false;
```

```
    } else if (year%4 == 0){
        return true;
    } else {
        return false;
    }
}

// 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;
    if (isLeapYear(year)) {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: 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;

        }
    } else {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: days = 28;
            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;
        }
```

```
    }
    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
  static int sundays = 0;
  /**
   * 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;
    //// 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");
        if (dayOfMonth == 1) {
          sundays++;
        }
      } else {
        System.out.println(dayOfMonth + "/" + month + "/" + year);
      }
      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 (debugDaysCounter == 36531) {
            break;
        }
    }
    System.out.println("During the 20th century, " + sundays + " 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() {
    dayOfMonth++;
    dayOfWeek = (dayOfWeek % 7) + 1;
    if (dayOfMonth > nDaysInMonth) {
        dayOfMonth = 1;
        month++;
        if (month > 12) {
            month = 1;
            year++;
        }
    }
    nDaysInMonth = nDaysInMonth(month,year);
  }

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

```
        return false;
    }
}

// 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 (isLeapYear(year)) {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: 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;

        }
    } else {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: days = 28;
            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;
        }
    }
    return days;
}
```

```
}
```

```
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
  static int sundays = 0;
  static boolean done = false;
  static int yearInput;
  /**
   * 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;
     yearInput = Integer.parseInt(args[0]);
     //// Write the necessary initialization code, and replace the condition
     //// of the while loop with the necessary condition
     while (true) {
        advance();
        if ((year == yearInput) || (debugDaysCounter == 36500)) {
           break;
        }
     }
     while (done) {
        if ((year == (yearInput+1)) || (debugDaysCounter == 366)) {
           break;
```

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

  // 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() {
    dayOfMonth++;
    dayOfWeek = (dayOfWeek % 7) + 1;
    if (dayOfMonth > nDaysInMonth) {
        dayOfMonth = 1;
        month++;
        if (month > 12) {
            month = 1;
            year++;
        }
        if (year == yearInput) {
            done = true;
        }
    }
    nDaysInMonth = nDaysInMonth(month,year);
  }
```

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

// 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 (isLeapYear(year)) {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: 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;

        }
    } else {
        switch (month) {
            case 1: days = 31;
            break;
            case 2: days = 28;
            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;
        }
    }
    return days;
  }
}
```