```
// /**
// * 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) {
      iterationCounter=0;
      double payment = loan/n;
      while (endBalance(loan, rate, n, payment)>=epsilon)
      {
         payment = payment + epsilon;
         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 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 bisectionSolver(double loan, double rate, int
n, double epsilon) {
      iterationCounter=0;
      double H= loan;
      double L = loan/n;
      double payment=(L+H)/2;
      while((H-L)>epsilon)
      {
           if (endBalance(loan, rate, n, payment) *endBalance(loan,
rate, n, L)>=0)
                L=payment;

           else
                H=payment;

           payment = (L + H) / 2;
           iterationCounter++;
      }

      return payment;
    }

      /**
      * 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) {

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

      }
}
```

```
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 answer = "";
        for (int i=0; i<s.length();i++)
        {
          char c = s.charAt(i);
          if (c>=65 && c<=90)
          {
                c=(char)(c+32);
          }
            answer=answer+c;
        }
        return answer;
    }
}
```

```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 answer="";
        for (int i=0; i<s.length();i++)
        {
            char c = s.charAt(i);
            if( c == 32|| answer.indexOf(c)==-1)
                answer=answer+c;
        }
        return answer;
    }
}
```

```
import java.time.DayOfWeek;

/**
 * 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 sundayCounter=0;
         //// Write the necessary initialization code, and replace the
condition
         //// of the while loop with the necessary condition

           while (year<2000) {
                //// Write the body of the while
                System.out.print(dayOfMonth+"/"+month+"/"+year);
                if(dayOfWeek==1)
                {
                      System.out.print(" Sunday");
                      if(dayOfMonth==1)
                           sundayCounter++;
                }
                System.out.println();
                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==36500) {
                //    break;
                //}
        }

           System.out.println("During the 20th century, "
+sundayCounter+ " 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()
      {
           if(month==12 && dayOfMonth==31)// if you made it to the end
of the year start a new one
           {
                year++;
                month=1;
                dayOfMonth=1;
           }
           else
           {

                if (dayOfMonth==nDaysInMonth)//if you made it to the
end of the month start a new one, else advance
                {
                     month++;
                     if (month==13)
                          month=1;
                     nDaysInMonth=nDaysInMonth(month, year);
                     dayOfMonth=1;

                }
                else
                     dayOfMonth++;
           }
           if(dayOfWeek==7)//if you made it to the end of the week,
start over
                dayOfWeek=1;
           else
```

```
            dayOfWeek++;

    }

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

    private static int nDaysInMonth(int month, int year)
     {
        int days = 31;
        switch (month)
        {
            case 4: days= 30 ;
            break;
            case 6: days= 30 ;
            break;
            case 9: days= 30 ;
            break;
            case 11: days= 30 ;
            break;
            case 2:
                if (isLeapYear(year))
                    days=29;
                else
                    days=28;
                break;
            }
        return days;
    }
}
```

```
/**
 * Prints the calendar of a given year
 */
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 checkyear= Integer.parseInt(args [0]);
        int sundayCounter=0;
        while (year<checkyear)
         {
            advance();
         }
         while  (year==checkyear)
         {
           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(month==12 && dayOfMonth==31)// if you made it to the end of
the year start a new one
        {
            year++;
            month=1;
            dayOfMonth=1;
        }
        else
        {

            if (dayOfMonth==nDaysInMonth)//if you made it to the end
of the month start a new one, else advance
            {
                month++;
                if (month==13)
                    month=1;
                nDaysInMonth=nDaysInMonth(month, year);
                dayOfMonth=1;

            }
            else
                dayOfMonth++;
        }
        if(dayOfWeek==7)//if you made it to the end of the week, start
over
            dayOfWeek=1;
        else
            dayOfWeek++;

    }

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

    private static int nDaysInMonth(int month, int year)
     {
        int days = 31;
        switch (month)
        {
            case 4: days= 30 ;
            break;
            case 6: days= 30 ;
            break;
            case 9: days= 30 ;
            break;
            case 11: days= 30 ;
            break;
            case 2:
                if (isLeapYear(year))
                    days=29;
                else
                    days=28;
                break;
            }
        return days;
    }
}
```