```
/**
* 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]);
		// double payment = Double.parseDouble(args[3]);

		// System.out.println("Loan sum = " + loan + ", interest rate = " +
rate + "%, periods = " + n + "endBalance: " + endBalance(loan,rate,n,payment));


		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 g = loan/n;
    while (endBalance(loan, rate,n,g) > epsilon ) {
        g = 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) {
     iterationCounter = 0;
     double L = 0 , H = loan, g = (L + H) / 2;
     while (H-L > epsilon){
     // Sets L and H for the next iteration
        if (endBalance(loan, rate,n,g) > 0)
        // the solution must be between g and H
        // so set L or H accordingly
                L = g;
        else
        // the solution must be between L and g
        // so set L or H accordingly
                H=g;
                // Computes the mid-value (ɔ• ‘”) for the next iteration

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

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

```
/** 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 ans = "";
        int i = 0;
        while (i < s.length()) {
         char ch = s.charAt(i);
         if ((ch >= 'A') && (ch <= 'Z')) {
            ans = ans  + (char) (s.charAt(i) + 32);
            //i++;
         }
         else {
            ans = ans + ch;
         }
         i++;

        }
                return ans;

    }
}
```

```
/** 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 ans ="" + s.charAt(0);
        int i = 1;
        char ch1 =  s.charAt(0);

        while (i < s.length()) {
            ch1 =  s.charAt(i);
            if(ans.indexOf(ch1)==-1 || ch1== 32)
                ans = ans+ch1;

            i++;
        }
         return ans;
     }

 }
```

```
/*
 * 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 daysInMonth = nDaysInMonth(month,year);
		for (int i =1; i<=12; i++){
			System.out.println("Month " + i + " has " +
nDaysInMonth(i,year) + " days");
		}
	}

	// Returns true if the given year is a leap year, false otherwise.
	public static boolean isLeapYear(int year) {
		boolean isLeapYear;
		// Checks if the year is divisible by 400
		isLeapYear = ((year % 400) == 0);
		// Then checks if the year is divisible by 4 but not by 100
		isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) !=
0));
       return isLeapYear;
	}

	// 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 daysInMonth = 0;
                if (month == 1 || month == 3 || month == 5 || month == 7 || month ==
8 || month == 10 || month == 12)
                        daysInMonth = 31;
                        else{
                                if(month == 4 || month == 6 || month ==9 || month ==
11)
                                        daysInMonth= 30;
                                else{
                                        if (isLeapYear(year))
                                                daysInMonth = 29;
                                        else
                                                daysInMonth = 28;
                                }

                        }

                return daysInMonth;
        }
}
```

```
/**
 * 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[]) {
                int counter = 0;
                // 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;

              // Starting the calendar on 1/1/1900
            dayOfMonth = 1;
            month = 1;
            year = 1900;
            dayOfWeek = 2;     // 1.1.1900 was a Monday
            nDaysInMonth = 31; // Number of days in January


            //// 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)
                                    counter ++;
                        }
                        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 (false) {
                                break;
                        }
        }
        System.out.println("During the 20th century, " + counter + " 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() {
                nDaysInMonth = nDaysInMonth(month,year);
          if (dayOfWeek == 7)
                dayOfWeek = 1;
          else
                dayOfWeek ++;

          if (nDaysInMonth == 31){
                if (dayOfMonth == 31){
                        dayOfMonth =1;
                        if(month == 12){
                                month = 1;
                                year ++;
                        }
                        else
                                month ++;
                }
                else
                        dayOfMonth ++;
          }

                if (nDaysInMonth == 30){
                if (dayOfMonth == 30){
                        dayOfMonth =1;
                        if(month == 12){
                                month = 1;
                                year ++;
```

```
                }
                else
                        month ++;
        }
        else
                dayOfMonth ++;
  }

if (nDaysInMonth == 29){
        if (dayOfMonth == 29){
                dayOfMonth =1;
                if(month == 12){
                        month = 1;
                        year ++;
                }
                else
                        month ++;
        }
        else
                dayOfMonth ++;
}

        if (nDaysInMonth == 28){
        if (dayOfMonth == 28){
                dayOfMonth =1;
                if(month == 12){
                        month = 1;
                        year ++;
                }
                else
                        month ++;
        }
        else
                dayOfMonth ++;
}


}
```

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

```
            boolean isLeapYear;
                // Checks if the year is divisible by 400
                isLeapYear = ((year % 400) == 0);
                // Then checks if the year is divisible by 4 but not by 100
                isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) !=
0));
        return isLeapYear;
          }

          // 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 daysInMonth = 0;
                if (month == 1 || month == 3 || month == 5 || month == 7 || month ==
8 || month == 10 || month == 12)
                        daysInMonth = 31;
                        else{
                                if(month == 4 || month == 6 || month ==9 || month ==
11)
                                        daysInMonth= 30;
                                else{
                                        if (isLeapYear(year))
                                                daysInMonth = 29;
                                        else
                                                daysInMonth = 28;
                                }

                        }

                return daysInMonth;
          }
}
```

```
/**
 * 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
        //static int givenYear = Integer.parseInt(args[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;

              // Starting the calendar on 1/1/1900
            dayOfMonth = 1;
            month = 1;
            year = 1900;
            dayOfWeek = 2;     // 1.1.1900 was a Monday
            nDaysInMonth = 31; // Number of days in January
            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)
            {
                    advance();
            }
                    while (year == givenYear ) {
                            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)
                    if (false) {
                            break;
                    }
        }
        //System.out.println("During the 20th century, " + counter + " 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() {
                    nDaysInMonth = nDaysInMonth(month,year);
            if (dayOfWeek == 7)
                    dayOfWeek = 1;
            else
                    dayOfWeek ++;

            if (nDaysInMonth == 31){
                    if (dayOfMonth == 31){
                            dayOfMonth =1;
                            if(month == 12){
                                    month = 1;
                                    year ++;
                            }
                            else
                                    month ++;
                    }
                    else
                            dayOfMonth ++;
            }

                    if (nDaysInMonth == 30){
                    if (dayOfMonth == 30){
                            dayOfMonth =1;
```

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

if (nDaysInMonth == 29){
        if (dayOfMonth == 29){
                dayOfMonth =1;
                if(month == 12){
                        month = 1;
                        year ++;
                }
                else
                        month ++;
        }
        else
                dayOfMonth ++;
}

        if (nDaysInMonth == 28){
        if (dayOfMonth == 28){
                dayOfMonth =1;
                if(month == 12){
                        month = 1;
                        year ++;
                }
                else
                        month ++;
        }
        else
                dayOfMonth ++;
}


}
```

```
   // Returns true if the given year is a leap year, false otherwise.
        private static boolean isLeapYear(int year) {
           boolean isLeapYear;
                // Checks if the year is divisible by 400
                isLeapYear = ((year % 400) == 0);
                // Then checks if the year is divisible by 4 but not by 100
                isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) !=
0));
       return isLeapYear;
        }

        // 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 daysInMonth = 0;
                if (month == 1 || month == 3 || month == 5 || month == 7 || month ==
8 || month == 10 || month == 12)
                        daysInMonth = 31;
                        else{
                                if(month == 4 || month == 6 || month ==9 || month ==
11)
                                        daysInMonth= 30;
                                else{
                                        if (isLeapYear(year))
                                                daysInMonth = 29;
                                        else
                                                daysInMonth = 28;
                                }

                        }

                return daysInMonth;
        }
}
```