```
/**
* 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 iterationCounterBr;
	static int iterationCounterBi;    // 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: " + iterationCounterBr);

		// 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: " + iterationCounterBi);
	}
```

```
        /**
        * 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 payment= loan / n;
      iterationCounterBr= 0;
         while (endBalance(loan, rate, n, payment) > 0) {
             payment += epsilon;
             iterationCounterBr++;
         }
         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 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)
{
```

```
        iterationCounterBi= 0;
        double L= loan / n;
        double H= loan;
        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;
            iterationCounterBi++;
      }
        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) + ((loan - payment) * (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) {

        int len= s.length();
        String backWords= "";
        for (int i = 0; i < len; i++) {
            if  ((s.charAt(i) >= 'A') && (s.charAt(i) <= 'Z')) {
                backWords += (char)(s.charAt(i) + 32);
            }
            else {
                 backWords += s.charAt(i);
            }
         }
        return backWords;
}
}
```

```
/** 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) {
       int len= s.length();
       String unique= "" + s.charAt(0);
       int uniLen= unique.length();
       for (int i= 0; i < len; i++) {
           if (uniqueChars(s.charAt(i), unique)) {
              unique += s.charAt(i);
           }

       }
       return unique;
    }

    public static boolean uniqueChars(char a, String s) {
       for (int i= 0; i < s.length(); i++)
       {
           if ((s.charAt(i) == a) && s.charAt(i) != ' ') {
               return false;
           }
       }
```

```
        return true;
    }
}
```

```
/*
 * 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.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) {
        if ((year % 400) == 0) {
            return true;
        }
        else if (((year % 4) == 0) && ((year % 100) != 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) {
      if (month == 2) {
                if (isLeapYear(year)) {
                        return 29;
                }
                else {
                        return 28;
                }
      }
      else if (month == 4 || month == 6 || month == 9 || month == 11) {
                return 30;
      }
      else {
            return 31;
      }
    }
}
```

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

```
                    else {
                        System.out.println(dayOfMonth + "/" + month + "/" +
year);
                    }
                    advance();
        }
        System.out.println("During the 20th century, " + sundays + " 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() {
                dayOfWeek = (dayOfWeek == 7) ? 1 : dayOfWeek + 1;
            if (dayOfMonth < nDaysInMonth) {
                dayOfMonth++;
            }
            else {
                dayOfMonth= 1;
                if (month < 12) {
                    month++;
                    nDaysInMonth= nDaysInMonth(month, year);
                }
                else {
                    month= 1;
                    nDaysInMonth= nDaysInMonth(month, year);
                    year++;
                }
            }
        }
```

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

```
}
```

```
/**
 * 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[]) {
                // 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.
            //// Write the necessary initialization code, and replace the condition
            //// of the while loop with the necessary condition
            int y= Integer.parseInt(args[0]);
            while (year < y) {
                advance();
            }
                while (year == y) {
                        if (dayOfWeek == 1) {
                                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
                        }
```

```
                else {
                        System.out.println(dayOfMonth + "/" + month + "/" +
year);
                }
                advance();
        }
            //// 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() {
                dayOfWeek = (dayOfWeek == 7) ? 1 : dayOfWeek + 1;
            if (dayOfMonth < nDaysInMonth) {
                dayOfMonth++;
            }
            else {
                dayOfMonth= 1;
                if (month < 12) {
                    month++;
                    nDaysInMonth= nDaysInMonth(month, year);
                }
                else {
                    month= 1;
                    nDaysInMonth= nDaysInMonth(month, year);
                    year++;
                }
            }
        }

    // Returns true if the given year is a leap year, false otherwise.
```

```
    private static boolean isLeapYear(int year) {
        if ((year % 400) == 0) {
            return true;
        }
        else if (((year % 4) == 0) && ((year % 100) != 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) {
      if (month == 2) {
                if (isLeapYear(year)) {
                        return 29;
                }
                else {
                        return 28;
                }
      }
      else if (month == 4 || month == 6 || month == 9 || month == 11) {
                return 30;
      }
      else {
            return 31;
      }
    }
}
```