```
public class LoanCalc {

    static double epsilon = 0.001; // The computation tolerance
(estimation error)
    static int iterationCounter; // Monitors the efficiency of
the calculation
    static int n; // Number of periods
    static double g; // Periodic payment (used within
calculations)
    static double loan; // Loan amount
    static double rate; // Periodic interest rate (as a
percentage)
    static double payment; // Calculated periodic payment
    static double endBalance; // Ending balance of the loan
    static double low; // Lower bound of the payment in bisection
search
    static double high; // Upper bound of the payment in
bisection search

    /**
     * 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) {
        g = loan / n;
        iterationCounter = 0;
        while (endBalance(loan, rate, n, g) > epsilon) {
            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;
        low = loan / n;
        high = loan;
        g = (low + high) / 2;
```

```
        // logic of the bisection search
        while ((high - low) > epsilon) {
            if (endBalance(loan, rate, n, g) * endBalance(loan,
rate, n, low) > 0) {
                low = g;
            } else {
                high = g;
            }
            g = (low + high) / 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) {
        endBalance = loan;
        for (int i = 0; i < n; i++) {
            endBalance = (endBalance - payment) * (1 + rate /
100);
        }
        return endBalance;
    }
}
```

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

```
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 noDuplicates = "";
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            boolean isDuplicate = false;

            // Check for duplicates
            for (int j = 0; j < noDuplicates.length(); j++) {
                if (c == noDuplicates.charAt(j)) {
                    isDuplicate = true;
                    break;
                }
            }

            // Add it if it's a space
            if (!isDuplicate || c == ' ') {
                noDuplicates += c;
            }
        }

        return noDuplicates;
    }
}
```

```
public class Calendar {
    static int currentMonth;
    static int currentDay;
    static int startYear;
    static int curretYear;
    static int endYear;
    static int currentDayOfTheWeek;
    static int nDaysInMonth;
    static boolean isLeapYear;
    static int nDays;

    public static void main(String args[]) {
        startYear = 1990;
        curretYear = Integer.parseInt(args[0]);
        endYear = curretYear + 1;
        advance();
    }

    // Advances the date (day, month, year) and the day-of-the-
week
    public static void advance() {
        currentDayOfTheWeek = 2;
        while (startYear < endYear) {
            currentMonth = 1;
            while (currentMonth <= 12) {
                currentDay = 1;
                while (currentDay <= nDaysInMonth(currentMonth,
startYear)) {
                    if (startYear == curretYear) {
                        if (currentDayOfTheWeek <= 7) {
                            System.out.print(currentDay + "/" +
currentMonth + "/" + startYear);
                            if (currentDayOfTheWeek == 1) {
                                System.out.print(" Sunday");
                                currentDay++;
                                currentDayOfTheWeek++;
                            } else {
                                currentDay++;
                                currentDayOfTheWeek++;
                            }
                            if (currentDayOfTheWeek > 7) {
                                currentDayOfTheWeek = 1;
                            }
                        }
                        System.out.println();
                    } else {
```

```
                        if (currentDayOfTheWeek <= 7) {
                            if ((currentDay == 1) &&
(currentDayOfTheWeek) == 1) {
                                currentDay++;
                                currentDayOfTheWeek++;
                            } else {
                                currentDay++;
                                currentDayOfTheWeek++;
                            }
                            if (currentDayOfTheWeek > 7) {
                                currentDayOfTheWeek = 1;
                            }
                        }
                    }
                }
                currentMonth++;
            }
            startYear++;
        }
    }

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

    // Returns the number of days in the given month and year
    private static int nDaysInMonth(int curMonth, int startYear)
{
        switch (curMonth) {
            case 1, 3, 5, 7, 8, 10, 12: // January, March, May,
July, August, October, and December
                nDays = 31;
                break;
            case 2: // February
                nDays = isLeapYear(startYear) ? 29 : 28;
                break;
            case 4, 6, 9, 11: // April, June, September, and
November
                nDays = 30;
                break;
            default:
```

```
                nDays = 0;
                System.out.println("Invalid month");
                break;
        }
        return nDays;
    }
}
```