

```

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)
    {
        double g = loan / n;
        while (endBalance(loan, rate, n, g) > 0) {
            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 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 hi = loan, lo = loan / n, g = (hi + lo) / 2;
    while (hi - lo > epsilon) {
        iterationCounter++;
        if (endBalance(loan, rate, n, g) * endBalance(loan, rate, n, lo) > 0) {
            lo = g;
            g = (hi + lo) / 2;
        }
        else {
            hi = g;
            g = (hi + lo) / 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) {
    double x = loan;
    for (int i = 0; i < n; i++) {
        x = (x - payment) * (rate / 100 + 1);
    }
    return x;
}
}

```

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

```

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 newString = "";
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == 32) newString += s.charAt(i);
            else {
                if (newString.indexOf(s.charAt(i)) == -1) {
                    newString += s.charAt(i);
                }
            }
        }
        return newString;
    }
}

```

```

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 occurred on the first day of the month during this period.
     */
    public static void main(String args[]) {
        int sunday = 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
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
                if (dayOfMonth == 1) sunday++;
            }
            else System.out.println(dayOfMonth + "/" + month + "/" + year);
            advance();
            /// If you want to stop the loop after n days, replace the condition of the
            /// if statement with the condition (debugDaysCounter == n)
        }
        System.out.println("During the 20th century, " + sunday + " 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 (dayOfWeek == 7) dayOfWeek = 1;
        else dayOfWeek++;
        if (dayOfMonth == nDaysInMonth(month, year)) {
            dayOfMonth = 1;
            month++;
            if (month == 13) {
                month = 1;
                year++;
            }
        }
        nDaysInMonth = nDaysInMonth(month, year);
    }
}

```

```

    }
    else dayOfMonth++;

}

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

private static int nDaysInMonth(int month, int year) {
    if (month == 2 && isLeapYear(year)) return 29;
    switch (month) {
        case 1:
            return 31;
        case 2:
            return 28;
        case 3:
            return 31;
        case 4:
            return 30;
        case 5:
            return 31;
        case 6:
            return 30;
        case 7:
            return 31;
        case 8:
            return 31;
        case 9:
            return 30;
        case 10:
            return 31;
        case 11:
            return 30;
        case 12:
            return 31;
    }
    return 0;
}
}

```