1. Loan Calculations

```
/**
 * 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]); // getting the loan
from the user
        double rate = Double.parseDouble(args[1]); // getting the rate
from the user
        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) {
        // Replace the following statement with your code
        iterationCounter = 0;
        double test_loan = loan;
        double search = loan / n;
        {
            while (test_loan > epsilon) {
                test_loan = endBalance(loan, rate, n, search);
                iterationCounter++;
                search += epsilon;
            }
            iterationCounter--;
        }
        return search;

    }

    /**
     * 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) {
        // Replace the following statement with your code
        double L = loan / n;
        double H = loan;
        double middle = (L + H) / 2;
        iterationCounter = 0;
        while (H - L > epsilon) {
            if (endBalance(loan, rate, n, (middle)) * endBalance(loan,
rate, n, (L)) > 0) {
                L = middle;
            } else {
                H = middle;
            }
            middle = (L + H) / 2;
            iterationCounter += 1;
        }
        return (middle);
    }

    /**
     * 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) {
        // Replace the following statement with your code
        double test_loan = loan;
        rate = rate / 100;
        for (int i = 0; i < n; i++) {
            test_loan = test_loan - payment;
            test_loan = test_loan * (1 + rate);
        }
        return test_loan;
    }
}
```

2. LowerCase

```
/** 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) {
        char rel_char;
        String new_string = "";
        for (int i = 0; i < s.length(); i++) {
            rel_char = s.charAt(i);
            if (rel_char >= 65 && rel_char <= 90) {
                rel_char = (char) (rel_char + 32);
            }
            new_string += rel_char;
        }
        // Replace the following statement with your code
        return new_string;
    }
}
```

3. **Unique Characters**

```
/** 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 new_string = "";
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ' ' || new_string.indexOf(s.charAt(i)) == -
1) {
                new_string = new_string + s.charAt(i);
            }
        }
        // Replace the following statement with your code
        return new_string;
    }
}
```

4. Calendar final

```
/**
 * 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 String isSunday = "";
    static int total_sundays = 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;
        int inputYear = Integer.parseInt(args[0]);
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (year <= inputYear) {
            //// Write the body of the while
            advance();
            debugDaysCounter++;
            if (year == inputYear) {
                System.out.println(dayOfMonth + "/" + month + "/" + year +
" " + isSunday);

            }
            //// 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;
            // }
        }
        //// 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++;
    dayOfMonth++;
    if (dayOfMonth > nDaysInMonth(month, year)) {
        dayOfMonth = 1;
        month += 1;
        if (month > 12) {
            month = 1;
            year++;
        }
    }
    if (dayOfWeek % 7 == 1) {
        isSunday = "Sunday";
        if (dayOfMonth == 1) {
            total_sundays++;
        }
    } else {
        isSunday = "";
    }
}

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

```
        }
        return n;
    }

}
```