```java
/**
* 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]);

        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);
        // Side effect: modifies the class variable iterationCounter.



    }
```

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

        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) {
        double h = loan;
        double L = loan / n;
        double payment = (L + h) / 2;
        iterationCounter = 0;
        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;
```

```
            iterationCounter ++;

        }
        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) {
        double balance = loan;
        for (int i = 0; i < n; i++){
            balance = (balance - payment)*(1+ rate/100);
        }

        return balance;
    }
}
```

```java
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

    public static String lowerCase(String s) {
     String str = "";
        for(int var3 = 0; var3 < s.length(); var3++) {
            char var2 = s.charAt(var3);
        if (s.charAt(var3) >= 'A' && s.charAt(var3) <= 'Z') {
            var2 = (char)(s.charAt(var3) + 32);
        }

        str = str + var2;
        }

        return str;
    }
}
```

```java
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }


    public static String uniqueChars(String s){
        char mid = (char)(s.charAt(0));
        String var1 = mid +  "";
        for (int i = 1; i < s.length(); i++){
            char var2 = s.charAt(i);
            boolean unique = true;
            for (int j = 0; j < var1.length(); j++){
                mid = var1.charAt(j);
            if (var2 == mid && mid != 32){
                unique = false;
                break;
            }
            }
        if (unique)
            var1 = var1 + var2;

        }




        return var1 ;
    }
}
```

```
public class Calendar0 {
    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);
        isLeapYearTest(year);
        nDaysInMonthTest(year);
    }


    private static void isLeapYearTest(int year) {
        String commonOrLeap = "common";
        if (isLeapYear(year)) {
            commonOrLeap = "leap";
        }
        System.out.println(year + " is a " + commonOrLeap + " year");
    }


    private static void nDaysInMonthTest(int year) {
        for (int month = 1; month < 13; month++){
             System.out.println("Month " + month + " has " +
nDaysInMonth(month, year) + " days");


        }

    }


    public static boolean isLeapYear(int year) {
     boolean isLeapYear = ((year % 400) == 0);
     isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) !=
0));
        return isLeapYear;
     }




    public static int nDaysInMonth(int month, int year) {
        if (month == 1 || month == 3 || month == 5 || month == 7 ||
month == 8 || month == 10 || month == 12){
            return 31;
        }
        if (month == 4 || month == 6 || month == 9 || month == 11){
            return 30;
        }
        else{
```

```
            if (isLeapYear(year) == true)return 29;
            else return 28;
        }
    }

}
```

```java
/**
 * 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.
        int debugDaysCounter = 37000;
        int daysCount = 0;
        int counter = 0; // Counts the Sundays that are first day of
the month
        while(year < 2000){
            month = 1;
            while(month < 13){
                dayOfMonth = 1;
                while(dayOfMonth <= nDaysInMonth(month, year)){
                    if(daysCount == debugDaysCounter) return; // For
testing
                    if(dayOfWeek == 1){
                        System.out.println(dayOfMonth + "/" + month +
"/" + year + " Sunday");
                        if(dayOfWeek == dayOfMonth){
                            counter++;
                        }
                    }
                    else{
                        System.out.println(dayOfMonth + "/" + month +
"/" + year);
                    }
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
```

```
                    }
                    else{
                        dayOfWeek++;
                    }
                    dayOfMonth++;
                    daysCount++;
                }
                month++;
            }
            year++;
        }
        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() {
        int counter = 0; // Counts the Sundays that are first day of
the month
        while(year < 2000){
            month = 1;
            while(month < 13){
                dayOfMonth = 1;
                while(dayOfMonth <= nDaysInMonth(month, year)){
                    if(dayOfWeek == 1){
                        System.out.println(dayOfMonth + "/" + month +
"/" + year + " Sunday");
                        if(dayOfWeek == dayOfMonth){
                            counter++;
                        }
                    }
                    else{
                        System.out.println(dayOfMonth + "/" + month +
"/" + year);
                    }
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
                    }
                    else{
                        dayOfWeek++;
                    }
                    dayOfMonth++;
                }
```

```
                month++;
            }
            year++;
        }
        System.out.println("During the 20th century, " + counter + "
Sundays fell on the first day of the month");
     }


    private static boolean isLeapYear(int year) {
        boolean isLeapYear = ((year % 400) == 0);
        isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100)
!= 0));
        return isLeapYear;
    }


    private static int nDaysInMonth(int month, int year) {
        if (month == 1 || month == 3 || month == 5 || month == 7 ||
month == 8 || month == 10 || month == 12){
            return 31;
        }
        if (month == 4 || month == 6 || month == 9 || month == 11){
            return 30;
        }
        else{
            if (isLeapYear(year) == true)return 29;
            else return 28;
        }
    }
}
```

```java
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[]) {
        // Runs over the days until a day before the wanted year
        int wantedYear = Integer.parseInt(args[0]);
        while(year < wantedYear){
            month = 1;
            while(month < 13){
                dayOfMonth = 1;
                while(dayOfMonth <= nDaysInMonth(month, year)){
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
                    }
                    else{
                        dayOfWeek++;
                    }
                    dayOfMonth++;
                }
                month++;
            }
            year++;
        }
        //Prints the calendar of the wanted year
        while(year < wantedYear + 1){
            month = 1;
            while(month < 13){
                dayOfMonth = 1;
                while(dayOfMonth <= nDaysInMonth(month, year)){
                    if(dayOfWeek == 1){
                        System.out.println(dayOfMonth + "/" + month +
"/" + year + " Sunday");
                    }
                    else{
                        System.out.println(dayOfMonth + "/" + month +
"/" + year);
                    }
```

```
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
                    }
                    else{
                        dayOfWeek++;
                    }
                    dayOfMonth++;
                }
                month++;
            }
            year++;
        }

    }

    // 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() {
        int counter = 0; // Counts the Sundays that are first day of
the month
        while(year < 2000){
            month = 1;
            while(month < 13){
                dayOfMonth = 1;
                while(dayOfMonth <= nDaysInMonth(month, year)){
                    if(dayOfWeek == 1){
                        System.out.println(dayOfMonth + "/" + month +
"/" + year + " Sunday");
                        if(dayOfWeek == dayOfMonth){
                            counter++;
                        }
                    }
                    else{
                        System.out.println(dayOfMonth + "/" + month +
"/" + year);
                    }
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
                    }
                    else{
                        dayOfWeek++;
                    }
                    dayOfMonth++;
                }
                month++;
            }
```

```
            year++;
        }
     }


    private static boolean isLeapYear(int year) {
        boolean isLeapYear = ((year % 400) == 0);
        isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100)
!= 0));
        return isLeapYear;
    }


    private static int nDaysInMonth(int month, int year) {
        if (month == 1 || month == 3 || month == 5 || month == 7 ||
month == 8 || month == 10 || month == 12){
            return 31;
        }
        if (month == 4 || month == 6 || month == 9 || month == 11){
            return 30;
        }
        else{
            if (isLeapYear(year) == true)return 29;
            else return 28;
        }
    }
}
```