```
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 ;
        double increment = 0.001;
        iterationCounter = 0;
        double f = endBalance ( loan, rate, n, g);
        while (f >= epsilon && f >= 0){
            g += increment;
            f = endBalance ( loan, rate, n, g);
            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) {
        double h = loan;
        double l = loan / n ;
        double g = (l+h)/2 ;
        double f = endBalance ( loan, rate, n, g);
        iterationCounter=0;
        while ((h - l ) > epsilon) {
            if (endBalance(loan, rate, n, g)*(endBalance(loan, rate, n
, l))>0) {
                l = g ;
            } else {
                h = g;
            }
            g = (l+h)/2 ;
            f= endBalance ( loan, rate, n, g);
            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) {

        for (int i = 0; i<n; i++ ) {
            loan = (loan - payment) * (0.01 * rate + 1);

        }
        return loan;
    }
}
```

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

            ans += c;
        }

        return ans;
    }
}
```

```java
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 ans = "";
        for ( int i=0; i < s.length(); i++){
            char c= s.charAt(i);
            if (s.indexOf(c)==i || c== 32){
                ans += c;
            }
    }
    return ans;
}
}
```

```java
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) {
        boolean isLeapYear;
        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.
    // 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) {
        switch (month){
            case 1:
                return 31;
            case 2:
                if (isLeapYear(year)){
                    return 29;
                } else{
                    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;
    }
}
```

```
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
    static int firstSunday = 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;
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (year < 2000) {
            if (dayOfMonth == 1 && dayOfWeek ==1){
                System.out.println(dayOfMonth + "/" + month +"/" + year
+ " Sunday");
                firstSunday ++;
            }else{
                System.out.println(dayOfMonth + "/" + month + "/"+
year);
            }
            advance();
            debugDaysCounter++;
            //// 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, " + firstSunday +
" 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() {
        nDaysInMonth = nDaysInMonth (month, year);
            if (dayOfWeek < 7){
                dayOfWeek++;
            }else{
                dayOfWeek =1;
            }
            if (dayOfMonth< nDaysInMonth){
                dayOfMonth ++;
            }else{
                dayOfMonth =1;
                if (month < 12){
                    month++;
                }else{
                    month=1;
                    year ++;
                }
            }
    }



    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        boolean isLeapYear;
        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.
    // 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) {
        switch (month){
            case 1:
                return 31;
            case 2:
                if (isLeapYear(year)){
                    return 29;
                } else{
                    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;
    }
}
```

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





     // 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() {
        nDaysInMonth = nDaysInMonth (month, year);
            if (dayOfWeek < 7){
                dayOfWeek++;
            }else{
                dayOfWeek =1;
            }
            if (dayOfMonth< nDaysInMonth){
                dayOfMonth ++;
            }else{
                dayOfMonth =1;
                if (month < 12){
                    month++;
                }else{
                    month=1;
                    year ++;
                }
            }
    }



    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        boolean isLeapYear;
        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.
    // 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) {
        switch (month){
            case 1:
                return 31;
            case 2:
                if (isLeapYear(year)){
                    return 29;
                } else{
                    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;
    }
}
```