```
/**
* 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);
	}

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

```
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++) {
            if(s.charAt(i) >= 65  && s.charAt(i) <= 90) {
                char c = (char)((int)s.charAt(i) + 32);
                lowerCaseString += c;
            }
            else
                lowerCaseString += s.charAt(i);
        }
        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 newStr = "";
        boolean chIsInside = false;
        for(int i = 0; i < s.length() ; i++) {
            for(int j = 0 ; j < newStr.length() ; j++){
                if(s.charAt(i) == newStr.charAt(j) && s.charAt(i) != ' ')
                    chIsInside = true;
            }
            if(!chIsInside)
                newStr += s.charAt(i);
            chIsInside = false;
        }
        return newStr;
    }
}
```

```
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[]) {
                int yearInput = Integer.parseInt(args[0]);
                while (year<yearInput) {
                        advance();
                }
                while(year == yearInput) {
                        if(dayOfWeek==1) {
                                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
                        }
                        else {
                                System.out.println(dayOfMonth + "/" + month + "/" + year);
                        }
                        advance();
                }
        }

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

```
    }

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

    // 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) {
        int daysNum = 0;
        switch (month) {
            case 1:
                daysNum = 31;
                break;
            case 2:
                if(isLeapYear(year))
                    daysNum = 29;
                else
                    daysNum = 28;
                break;
            case 3:
                daysNum = 31;
                break;
            case 4:
                daysNum = 30;
                break;
            case 5:
                daysNum = 31;
                break;
            case 6:
                daysNum = 30;
                break;
            case 7:
                daysNum = 31;
                break;
            case 8:
                daysNum = 31;
                break;
            case 9:
                daysNum = 30;
                break;
            case 10:
                daysNum = 31;
                break;
            case 11:
                daysNum = 30;
                break;
            case 12:
                daysNum = 31;
```

```
                        break;
                }
                return daysNum;
        }
}
```