Loan calc:

```
/**
* 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 g = loan/n ;
		while (endBalance(loan,rate,n,g) >= epsilon){
			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 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 L = 0 ;
		double H = loan ;
		double g = (L + H)/2 ;
		while ((H-L) >= epsilon){
			if ((endBalance(loan,rate,n,g))*(endBalance(loan,rate,n,H))  < 0 ){
				L = g ;
```

```
                }else{
                        H = g ;
                }
                g = (L + H)/2 ;
                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) {

        double balance = 0 ;
        double currentBalance = loan ;

        for(int i =1 ; i <= n ; i++){

                balance = (currentBalance - payment)*(1+(rate/100)) ;
                currentBalance = balance ;
        }

    return balance ;
    }
}
```

Lower case:

```
/** 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) {

      String answer = "" ;

      for(int i = 0 ; i < s.length() ; i++){
         char letter = s.charAt(i) ;

         if ((letter >= 'A') && (letter <= 'Z') ){
            answer = answer + (char) (letter + 32) ;
         }else{
            answer = answer + letter ;
         }
      }

      return answer ;
   }
}
```

Unique cahars:

```
/** 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 answer = "" ;

      for(int i = 0 ; i < s.length() ; i++){

         char letter = s.charAt(i) ;

         if (letter == ' ' || answer.indexOf(letter) == -1){
            answer = answer + letter ;
         }
      }



      return answer ;
   }
}
```

Calendar:

```
/**
 * 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 int specialSundaysCounter = 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 currentYear = Integer.parseInt(args[0]);
                while (year < currentYear) {

                        while(month <13){

                                nDaysInMonth = nDaysInMonth(month,year) ;
                                for(int j = 1; j<= nDaysInMonth; j++){
                                        if(dayOfWeek ==7){
                                                dayOfWeek =1 ;
                                        }else{
                                                dayOfWeek++ ;
                                        }
                                }
                                month++ ;
                        }
                        year++ ;
```

```
                month =1;
        }
        month = 1;
        for(int i=1 ; i<=12 ; i++){

                advance(dayOfMonth,month,year);
                month++ ;
        }
                if (month == 13){
                        month =1 ;
                }
                year++ ;
}




        //// 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 dayOfMonth, int month, int year) {

        for(dayOfMonth= 1 ; dayOfMonth <= nDaysInMonth(month,year) ; dayOfMonth++){

                System.out.print(dayOfMonth + "/" + month + "/" + year) ;

                if(dayOfWeek == 1){
                        System.out.print(" Sunday") ;

                }
                if(dayOfWeek == 7){
```

```
                dayOfWeek = 1 ;
            }else{
                dayOfWeek++ ;

            }

            System.out.println() ;

        }




    }

// Returns true if the given year is a leap year, false otherwise.
    public static boolean isLeapYear(int year) {

        return (((year % 400) == 0) || (((year % 4) == 0) && ((year % 100) != 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.
    public static int nDaysInMonth(int month, int year) {

        int daysInMonth = 0;


        if(month == 4 || month == 6 || month == 9 || month == 11){
            daysInMonth = 30;
```

```
                }else if (month == 1 || month == 3 || month == 5 ||month == 7 || month == 8 || month == 10 ||
month == 12){
                        daysInMonth = 31;
                }else if (month == 2 && isLeapYear(year)){
                        daysInMonth = 29;
                }else{
                        daysInMonth = 28;
                }

                return daysInMonth;
        }
}

/*for(dayOfWeek =2 ; dayOfWeek <= 7 ; dayOfWeek)
                for(dayOfMonth= 1 ; dayOfMonth <= nDaysInMonth(month,year) ; dayOfMonth++){
                        if(dayOfWeek=1){
                                System.out.println(dayOfMonth + "/" + month + "/" + year + "sunday")
                        }else{
                                System.out.println(dayOfMonth + "/" + month + "/" + year)
                        }


                }
                for(month = 1 ; month <= 12 ; month++)
                for(year = 1900; year <2000 ; year++)
                if(dayOfWeek=1 && dayOfMonth= 1 ){
                        firstSundaysCounter++ ;
                }*/
```