1. LoanCalc

```
/**
* 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=0;    // 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);
                iterationCounter=0;
                // 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
        double g=loan/n;
        while(endBalance(loan,rate,n,g)>0){
                        g=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)
{
        // Replace the following statement with your code
```

```
        double l=loan/n;
        double h=loan;
        double g=(l+h)/2;

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

2. 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) {
        // Replace the following statement with your code
            String newStr="";
              for(int i = 0; i<s.length(); i++){
                  char c= s.charAt(i);
                     if(c>='A' && c<='Z'){
                        c=((char)(c+32));
                     }
                  newStr+=c;
              }
        return newStr;
    }
}
```

3. UniqueChars

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

4. Calendar0

```
/*
 * Checks if a given year is a leap year or a common year,
 * and computes the number of days in a given month and a given year.
 */
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<13; 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) {
		if(((year % 400) == 0) || (((year % 4) == 0) && ((year % 100) != 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.
        public static int nDaysInMonth(int month, int year) {
            int days;
            if(month==2){
                if (isLeapYear(year)){
                        days= 29;
                    }
                        else {
                            days= 28;
                        }
                }
                else {  if ( (month==4) || (month==6) || (month==9) ||
(month==11) ){
                        days=30;
                }
                        else{
                            days=31;
                        }
                }
            return days;
        }
}
```

5. Calendar1

```
/**
 * 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 sundayCounter = 0;
            //// Write the necessary initialization code, and replace the condition
            //// of the while loop with the necessary condition
                while (year<=1999) {
                        System.out.print(dayOfMonth+ "/" +month+ "/" +year);

                        if (dayOfWeek==1){
```

```
                        System.out.println(" Sunday");
                        if(dayOfMonth==1){
                                sundayCounter++;
                        }
                }
        else{
                System.out.println();
        }

                advance();
        }

                System.out.println("During the 20th century, "+sundayCounter+" 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() {
                // Replace this comment with your code
                if(dayOfMonth < nDaysInMonth(month,year)) {
                        dayOfMonth++;
                } else {
                        month++;
                        dayOfMonth = 1;
                }
                if(month > 12) {
                        month = 1;
                        year++;
```

```
            }
            if(dayOfWeek % 7 != 0) {
                        dayOfWeek++;
            } else {
                        dayOfWeek = 1;
            }
      }


// 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 % 400) == 0) || (((year % 4) == 0) && ((year % 100) != 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 days;
                if(month==2){
                     if (isLeapYear(year)){
                                 days= 29;
```

```
                        }
                                else {
                                        days= 28;
                                }
                        }
                        else {
                                if ( (month==4) || (month==6) || (month==9) ||
(month==11) ){
                                        days=30;
                        }
                                        else{
                                                days=31;
                                        }
                        }

                return days;
        }
}
```

6. 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

        /**
         * 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 givenYear = Integer.parseInt(args[0]);
            int sundayCounter = 0;

            while (year<givenYear){
                advance();
            }
            //// Write the necessary initialization code, and replace the condition
            //// of the while loop with the necessary condition
```

```
            while (year==givenYear) {
                System.out.print(dayOfMonth+ "/" +month+ "/" +givenYear);
                if (dayOfWeek==1){
                    System.out.println(" Sunday");
                }
      else{
            System.out.println();
      }
          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() {
            // Replace this comment with your code
            if(dayOfMonth < nDaysInMonth(month,year)) {
                dayOfMonth++;
            } else {
                month++;
                dayOfMonth = 1;
            }
            if(month > 12) {
                month = 1;
                year++;
            }
            if(dayOfWeek % 7 != 0) {
                    dayOfWeek++;
```

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


// 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 % 400) == 0) || (((year % 4) == 0) && ((year % 100) != 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 days;
                if(month==2){
                    if (isLeapYear(year)){
                                days= 29;
                        }
                                else {
                                        days= 28;
```

```
                    }
                }
                else {
                    if ( (month==4) || (month==6) || (month==9) ||
(month==11) ){
                        days=30;
                }
                        else{
                            days=31;
                        }
                }

            return days;
        }
}
```