Calendar0

```
public class Calendar0 {
        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){
          if ((year % 400 == 0) || (year % 4 == 0) && (year % 100 !=0)){
             return true;
           }
        return false;
    }
    public static int nDaysInMonth(int month, int year) {
        int days;
        if(month == 4 || month == 6 || month ==9 || month ==11)
            days = 30;
        else if(month == 2){
            if(isLeapYear(year)){
                days = 29;
            } else
                days = 28;
        } else
            days = 31;
        return days;
      }
    }
```

Calendar1

```
public class Calendar1 {
	static int dayOfMonth = 1;
	static int month = 1;
	static int year = 1990;
	static int dayOfWeek = 2;          // 1.1.1900 was a Monday
	static int nDaysInMonth = 31; // Number of days in January

	public static void main(String args[]) {
		 int debugDaysCounter = 0;
		int Sundays = 0;
		while (year != 2000 ) {
			if(dayOfWeek == 1 && dayOfMonth == 1){
					System.out.println(dayOfMonth + "/" + month + "/" + year + "
Sunday");
			} else {
				System.out.println(dayOfMonth + "/" + month + "/" + year);
			}
			if(dayOfWeek == 1){
				Sundays++;
			}
			advance();
			debugDaysCounter++;
			}
			System.out.println("During the 20th century, " + Sundays + " Sundays
fell on the first day of the month");
		}
	private static void advance() {
			if (dayOfWeek == 7){
				dayOfWeek = 1;
			} else
				dayOfWeek++;
			if(dayOfMonth == nDaysInMonth(month, year) && month != 12){
				month ++;
				dayOfMonth = 1;
			}
			else if(dayOfMonth == nDaysInMonth(month, year) && month == 12){
				year++;
				dayOfMonth = 1;
				month = 1;
		} else
				dayOfMonth++;
	}
```

```
private static boolean isLeapYear(int year) {
       if ((year % 400 == 0) || (year % 4 == 0) && (year % 100 !=0)){
                return true;
      }
               return false;
}
private static int nDaysInMonth(int month, int year) {
                 int days;
           if(month == 4 || month == 6 || month ==9 || month ==11)
              days = 30;
      else if(month == 2){
                if(isLeapYear(year)){
                  days = 29;
             } else
                days = 28;
      } else
             days = 31;
      return days;
}

}
```

Calendar

```
public class Calendar {
        static int year = 1900;
      static int dayOfMonth = 1;
      static int month = 1;
      static int dayOfWeek = 2;
      static int nDaysInMonth = 31;

      public static void main(String args[]) {
            int yearA = Integer.parseInt(args[0]);
             while (year <= yearA ){
                 advance();
                if (year == yearA) {
                     if(dayOfWeek == 1){
                                System.out.println(dayOfMonth + "/" + month + "/" + year +
" Sunday");
                     } else {
                                System.out.println(dayOfMonth + "/" + month + "/" + year);

                 }
                 }
           }
             }

       private static void advance() {
           if (dayOfWeek == 7){
                 dayOfWeek = 1;
           } else
                  dayOfWeek++;
           if(dayOfMonth > nDaysInMonth(month, year)){
                 if(month < 12){
                   month ++;
                   dayOfMonth = 1;
                 } else{
                 year++;
                 dayOfMonth = 1;
                 month = 1;
                 }
            } else
            dayOfMonth++;
       }

      private static boolean isLeapYear(int year) {
            if ((year % 400 == 0) || (year % 4 == 0) && (year % 100 !=0)){
```

```
            return true;
        } else
          return false;
}

private static int nDaysInMonth(int month, int year) {
        int nDaysInMonth;
        if(month == 4 || month == 6 || month ==9 || month ==11)
             nDaysInMonth = 30;
        else if(month == 2){
              if(isLeapYear(year)){
                    nDaysInMonth = 29;
              } else
                 nDaysInMonth = 28;
        } else
               nDaysInMonth = 31;
        return nDaysInMonth;
}

}
```

LowerCase

```
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 str) {
		String lowerCase = "";
	    for (int i = 0; i < str.length(); i++){
	      if((str.charAt(i) >= 'A')&&(str.charAt(i) <= 'Z')){
			lowerCase += (char) (str.charAt(i)+32);

		  } else {
			lowerCase += str.charAt(i);
		  }
		}
		  return lowerCase;
	}
}
```

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 str) {
		   String uniqueChars = "";
		boolean duplicate;
		int n;
	   for (int i = 0; i < str.length(); i++ ){
		 duplicate = false;
		if(str.charAt(i) != ' '){
		     n = str.indexOf(str.charAt(i));
		     if(str.charAt(n) == str.charAt(i) && (i != n)){
		     duplicate = true;
		     }
		    }
		if (!duplicate)
		       uniqueChars += str.charAt(i);
	   }
		return uniqueChars;
	}
}
```

LoanCalc

```
public class LoanCalc {
    static double epsilon = 0.001;    // The computation tolerance (estimation error)
    static int iterationCounter;      // Monitors the efficiency of the calculation
    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]);
        rate = (double)rate / 1.0;
        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.
    */
    public static double bisectionSolver(double loan, double rate, int n, double epsilon) {
        double low = loan / n;
        double high = loan;
        double g = 0;
        iterationCounter = 0;
        while(high-low > epsilon){
            g = (low + high) / 2;
            double end = endBalance(loan, rate, n, g);
            if (Math.abs(end) <= epsilon){
                break;
            } else if (end > 0){
                low = g;
            }else{
                high = 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) {
        double currentloan = loan;
        double nextBal = 0;
        for (int i =0; i<n; i++){
            nextBal = (currentloan -payment) * (1 + rate / 100);
            currentloan = nextBal;
        }
        return currentloan;
}
}
```