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;    // 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;
        iterationCounter=0;
        double increment = 0.0001;//can be changed to any increment smaller than epsilon
        while((endBalance(loan, rate, n, g)) >= epsilon)
        {
            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) {
        iterationCounter=0;
        double l = loan/n;
        double h = loan;
        double g = (h + l)/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) {
```

```
        double endLoan =loan;
        for (int i = n; i>0; i--)
        {
            endLoan = (endLoan-payment)*(1.0+rate/100);
        }

        return endLoan;
    }
}
```

LowerCase

```
/** 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 newWord = "";
        for(int i = 0; i<s.length(); i++)
        {
            if(s.charAt(i) > 64 && s.charAt(i) < 91)
            {
               newWord = newWord + (char)(s.charAt(i) + 32);
            }
            else{
                newWord = newWord + s.charAt(i);
            }
        }
        return newWord;
    }
}
```

UniqueChars

```
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));

    }

    public static String uniqueChars(String s)
    {
        String newWord = "";
        for(int i=0; i<s.length(); i++)
        {
            if(s.charAt(i) == ' ')
            {
                newWord = newWord + " ";
            }
            if(newWord.indexOf(s.charAt(i)) == -1)
            {
                newWord = newWord + s.charAt(i);
            }

        }
        return newWord;
    }
    /**
     * 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 newWord= "" + s.charAt(0);
        for(int i=0;i< s.length()-1; i++)
        {
            for(int j=0; j<newWord.length();j++)
            {

                if (newWord.charAt(j)==s.charAt(i)) {
                j=newWord.length()+10;
                }
                if(j==newWord.length()-1)
```

```
            {
                newWord=newWord+s.charAt(i);
            }
            if(s.charAt(i)==' ')
            {
                newWord=newWord+s.charAt(i);
                j=newWord.length()+10;
            }

        }
    }
    //
    //
    //check if charat(i)  exist in the new word with ---> indexOn('')
    //
    //
    //need to check this- for repetition it works.
    if(newWord.charAt(newWord.length()-1)!=s.charAt(s.length()-1))
    {
        newWord= newWord + s.charAt(s.length()-1);
    }
   return newWord;
  }*/
}
```

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 sundayCounter=0;//count sundays on the first's

   public static void main(String args[]) {
      int debugDaysCounter = 0;
      int yearToPrint = Integer.parseInt(args[0]);
      while (year <= yearToPrint) {
         if (year == yearToPrint) {
            if (dayOfWeek == 1) {
             System.out.println(dayOfMonth+"/"+month+"/"+year + " Sunday");
             }
             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)
         if (false) {
            break;
         }
      }
      //System.out.println("During this year , "+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() {
```

```
if(nDaysInMonth(month, year) == 31){
    if(dayOfMonth<31)
    {
        dayOfMonth++;
    }
    else{
        dayOfMonth = 1;
        month++;
        if (month==13) {
            month=1;
        }
    }
}
else if(nDaysInMonth(month, year) == 30){
    if(dayOfMonth<30)
    {
        dayOfMonth++;
    }
    else{
        dayOfMonth = 1;
        month++;
    }
}
else if(nDaysInMonth(month, year) == 29){
    if(dayOfMonth<29)
    {
        dayOfMonth++;
    }
    else{
        dayOfMonth = 1;
        month++;
    }
}
    else if(nDaysInMonth(month, year) == 28){
    if(dayOfMonth<28)
    {
        dayOfMonth++;
    }
    else{
        dayOfMonth = 1;
        month++;
    }
}


if(month == 1 && dayOfMonth==1)
```

```
        {
            year++;
        }

        if (dayOfWeek%7 == 0) {
            dayOfWeek = 1;
            if(dayOfMonth == 1){
                    sundayCounter++;
            }
        }
        else{
            dayOfWeek++;
        }
    }
// Returns true if the given year is a leap year, false otherwise.
public static boolean isLeapYear(int year) {
    boolean leapOrNot = ((year % 400) == 0);
    leapOrNot = leapOrNot || (((year % 4) == 0) && ((year % 100) != 0));
    return leapOrNot;
}

// 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 numOfDays = 0;
    if(isLeapYear(year))//it is a leap year
    {
        switch (month) {
            case 1: numOfDays = 31;
                    break;
            case 2: numOfDays = 29;
                    break;
            case 3: numOfDays = 31;
                    break;
            case 4: numOfDays = 30;
                    break;
            case 5: numOfDays = 31;
                    break;
            case 6: numOfDays = 30;
                    break;
            case 7: numOfDays = 31;
                    break;
            case 8: numOfDays = 31;
                    break;
```

```
            case 9: numOfDays = 30;
                    break;
            case 10: numOfDays = 31;
                    break;
            case 11: numOfDays = 30;
                    break;
            case 12: numOfDays = 31;
                    break;
            default: numOfDays = -1;
                break;
        }
    }
    else
    {
        switch (month) {
            case 1: numOfDays = 31;
                    break;
            case 2: numOfDays = 28;
                    break;
            case 3: numOfDays = 31;
                    break;
            case 4: numOfDays = 30;
                    break;
            case 5: numOfDays = 31;
                    break;
            case 6: numOfDays = 30;
                    break;
            case 7: numOfDays = 31;
                    break;
            case 8: numOfDays = 31;
                    break;
            case 9: numOfDays = 30;
                    break;
            case 10: numOfDays = 31;
                    break;
            case 11: numOfDays = 30;
                    break;
            case 12: numOfDays = 31;
                    break;
            default: numOfDays = -1;
                break;
        }
    }
    return numOfDays;
  }
}
```

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
    static int sundayCounter=0;//count sundays on the first's
    /**
     * 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 <= 1999) {
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth+"/"+month+"/"+year + " Sunday");
            }
            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)
            if (false) {
                break;
            }
        }
        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() {

        if(nDaysInMonth(month, year) == 31){
            if(dayOfMonth<31)
            {
                dayOfMonth++;
            }
            else{
                dayOfMonth = 1;
                month++;
                if (month==13) {
                    month=1;
                }
            }
        }
        else if(nDaysInMonth(month, year) == 30){
            if(dayOfMonth<30)
            {
                dayOfMonth++;
            }
            else{
                dayOfMonth = 1;
                month++;
            }
        }
        else if(nDaysInMonth(month, year) == 29){
            if(dayOfMonth<29)
            {
                dayOfMonth++;
            }
            else{
                dayOfMonth = 1;
                month++;
            }
        }
            else if(nDaysInMonth(month, year) == 28){
            if(dayOfMonth<28)
            {
                dayOfMonth++;
            }
            else{
```

```
                dayOfMonth = 1;
                month++;
            }
        }


        if(month == 1 && dayOfMonth==1)
        {
            year++;
        }

        if (dayOfWeek%7 == 0) {
            dayOfWeek = 1;
            if(dayOfMonth == 1){
                    sundayCounter++;
            }
        }
        else{
            dayOfWeek++;
        }
    }



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

// 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 numOfDays = 0;
    if(isLeapYear(year))//it is a leap year
    {
        switch (month) {
            case 1: numOfDays = 31;
                    break;
            case 2: numOfDays = 29;
                    break;
            case 3: numOfDays = 31;
                    break;
```

```
        case 4: numOfDays = 30;
                break;
        case 5: numOfDays = 31;
                break;
        case 6: numOfDays = 30;
                break;
        case 7: numOfDays = 31;
                break;
        case 8: numOfDays = 31;
                break;
        case 9: numOfDays = 30;
                break;
        case 10: numOfDays = 31;
                break;
        case 11: numOfDays = 30;
                break;
        case 12: numOfDays = 31;
                break;
        default: numOfDays = -1;
           break;
    }
}
else
{
    switch (month) {
        case 1: numOfDays = 31;
                break;
        case 2: numOfDays = 28;
                break;
        case 3: numOfDays = 31;
                break;
        case 4: numOfDays = 30;
                break;
        case 5: numOfDays = 31;
                break;
        case 6: numOfDays = 30;
                break;
        case 7: numOfDays = 31;
                break;
        case 8: numOfDays = 31;
                break;
        case 9: numOfDays = 30;
                break;
        case 10: numOfDays = 31;
                break;
        case 11: numOfDays = 30;
```

```
                break;
            case 12: numOfDays = 31;
                break;
            default: numOfDays = -1;
                break;
        }
    }
    return numOfDays;
  }
}
```