HW 3 - Shahar Horovitz

```
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
		iterationCounter = 0;
		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 payment = loan/n;
	double increment = 0.001;
	while (endBalance(loan,rate,n,payment) > epsilon)
	{
		payment +=increment;
		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 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 bisectionSolver(double loan, double rate, int n, double epsilon) {
        double H = loan;
        double L = 0;
      double g = (L+H)/2.0;
                while ((H - L) > epsilon) {
        g = (L+H)/2.0;
                        if (endBalance(loan,rate,n,g) * endBalance(loan, rate, n, L)<0)
                        {
                                H = g;
                        }
                else
                {
                        L = 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) {
                //function should return thr end balance of the loan
                double annual_rate = 1+ (rate/100);
                for (int i =0; i<n; i++) //enters a for loop for the number of payments
                {
                        loan = (loan - payment) * annual_rate;
                }
                return loan;

        }
}
```

```
public class LowerCase {
    public static void main(String[] args)
    {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

    public static String lowerCase(String s) {
    String only_lower = "";
    for(int i =0; i<s.length(); i++){ //runs on the the string to check each charcter
            if(!Character.isDigit(s.charAt(i))){ //if the charcter is not a digit
                    only_lower = only_lower + Character.toLowerCase(s.charAt(i)); //make it lower case
            }
            else {
                    only_lower = only_lower + s.charAt(i); //if it's a digit, leave it
            }
    }
        return only_lower;
    }
}


/** 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 new_string = "";
        String seen_char = "";
    for(int i =0; i<s.length(); i++)
    { //runs on the the string to check each charcter
            char currentchar = s.charAt(i);
            if(currentchar == ' ')
            {
                    new_string = new_string + " ";
            }
            else if (seen_char.indexOf(currentchar) == -1)
            {
                    seen_char = seen_char + currentchar;
                    new_string = new_string + currentchar;
            }

    }
            return new_string;
        }
    }


/*
 * 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 %4 ==0)&& (year %100 != 0 || year %400 ==0))
	   {
		return true;
	   }
		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) {
		// Replace the following statement with your code
		if (month ==2){ //checks the month of february
			if(isLeapYear(year)){
				return 29; //if it's a leap year then return 29
			}
			else {
				return 28; //if not a leap year then return 28;
			}
		}
		if (month %2 ==0 && month >7)
		{
			return 31;
		}
		if (month % 2 == 0 && month < 7)
		{
			return 30;
		}
		if (month %2 ==1 && month <=7)
```

```
                {
                        return 31;
                }
                else {
                        return 30;
                }
        }
}


/**
 * 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 debugDaysCounter = 0;
            //// Write the necessary initialization code, and replace the condition
            //// of the while loop with the necessary condition
                int sundays = 0;
                while (year != 2000)
                {
                        //// Write the body of the while
                        sundays+=advance(dayOfMonth, month, year);
                        year++;
                        //debugDaysCounter++;
                        //// If you want to stop the loop after n days, replace the condition of the
                        //// if statement with the condition (debugDaysCounter == n)
                        //if (debugDaysCounter==1) {
                                //break;
                        //}
        }
                System.out.println("During the 20th century, " + sundays + " Sundays fell on the
first day of the month");
                //// 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 int advance(int day, int month, int year) {
		 int sundays = 0;
		for(int i =1; i<=12; i++)
		{
			for (int j = 1; j<=nDaysInMonth(i,year); j++)
			{
				if (dayOfWeek == 1)
				{
					if (j ==1)
					{
						sundays++;
					}
					//System.out.println(j + "/" +month +"/" + year + " Sunday");
					dayOfWeek++;
				}
				else if (dayOfWeek ==7)
				{
					//System.out.println(j + "/" +month +"/" + year);
					dayOfWeek=1;
				}
				else
				{
					//System.out.println(j + "/" +month +"/" + year);
					dayOfWeek++;
				}

			}
			month++;
		}
		year++;
		 return sundays;
}


   // Returns true if the given year is a leap year, false otherwise.
	private static boolean isLeapYear(int year) {
	   if ((year %4 ==0)&& (year %100 != 0 || year %400 ==0))
	   {
		return true;
	   }
		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) { //function returns the number of days
in a month
		if (month ==2){ //checks the month of february
			if(isLeapYear(year)){
				return 29; //if it's a leap year then return 29
			}
			else {
				return 28; //if not a leap year then return 28;
			}
		}
		if (month %2 ==0 && month >7)
		{
			return 31;
```

```
                }
                if (month % 2 == 0 && month < 7)
                {
                        return 30;
                }
                if (month %2 ==1 && month <=7)
                {
                        return 31;
                }
                else {
                        return 30;
                }
        }
}



/**
 * 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, useds for debugging purposes, counts how many days were
advanced so far.
            int current_year = Integer.parseInt(args[0]);
            int debugDaysCounter = 0;
            //// Write the necessary initialization code, and replace the condition
            //// of the while loop with the necessary condition
      int sundays = 0;
                while (year != 2030)
                {
                        //// Write the body of the while
                        sundays += advance(dayOfMonth, month, year, current_year);
        year++;
                        //debugDaysCounter++;
                        //// If you want to stop the loop after n days, replace the condition of the
                        //// if statement with the condition (debugDaysCounter == n)
                        if (debugDaysCounter==1)
        {
                                break;
                        }
      }
      //System.out.println("During the 20th century " + sundays + " sundays fell on the first day of
the month");
                //// 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 int advance(int day, int month, int year, int print_year) {
		int sundays = 0;
		if (year == print_year)
		{
			for(int i =1; i<=12; i++)
			{
			for (int j = 1; j<=nDaysInMonth(i,year); j++)
			{
				if (dayOfWeek == 1)
				{
					if (j ==1)
					{
						sundays++;
					}
					System.out.println(j + "/" +month +"/" + year + " Sunday");
					dayOfWeek++;
				}
				else if (dayOfWeek ==7)
				{
					System.out.println(j + "/" +month +"/" + year);
					dayOfWeek=1;
				}
				else
				{
					System.out.println(j + "/" +month +"/" + year);
					dayOfWeek++;
				}

			}
			month++;
		}
		year++;
		}
		else
			{
				for(int i =1; i<=12; i++)
		{
			for (int j = 1; j<=nDaysInMonth(i,year); j++)
			{
				if (dayOfWeek == 1)
				{
					if (j ==1)
					{
						sundays++;
					}
					//System.out.println(j + "/" +month +"/" + year + " Sunday");
					dayOfWeek++;
				}
				else if (dayOfWeek ==7)
				{
					//System.out.println(j + "/" +month +"/" + year);
					dayOfWeek=1;
				}
				else
```

```
                                {
                                        //System.out.println(j + "/" +month +"/" + year);
                                        dayOfWeek++;
                                }

                        }
                        month++;
                }
                year++;
        }
      return sundays;
}


   // Returns true if the given year is a leap year, false otherwise.
        private static boolean isLeapYear(int year) {
           if ((year %4 ==0)&& (year %100 != 0 || year %400 ==0))
           {
                return true;
           }
                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) { //function returns the number of days
in a month
                if (month ==2){ //checks the month of february
                        if(isLeapYear(year)){
                                return 29; //if it's a leap year then return 29
                        }
                        else {
                                return 28; //if not a leap year then return 28;
                        }
                }
                if (month %2 ==0 && month >7)
                {
                        return 31;
                }
                if (month % 2 == 0 && month < 7)
                {
                        return 30;
                }
                if (month %2 ==1 && month <=7)
                {
                        return 31;
                }
                else {
                        return 30;
                }
        }
}
```