Neta Yaar 208320010 HW3 Code

```
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 f = endBalance(loan, rate, n, g);
    while(f > epsilon){
        if (f > 0){
            g = g + epsilon;
        }

        iterationCounter++;
        f = endBalance(loan, rate, n, g);
    }
    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) {
    double low = loan/n;
    double high = loan;
    double g = ((low + high)/2);
    iterationCounter = 0;


    while ((high-low)> epsilon){
        double f = endBalance(loan, rate, n, g);
        double fL = endBalance(loan, rate, n, low);

        if (f * fL > 0){
            low = g;
        }
        else{
            high = g;
        }
        g= ((low + high)/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 = loan;
        rate /= 100;
        for (int i = n; i > 0; i--) {
            balance = ((balance - payment)*(1 + rate));
        }

        return balance;
    }
}
```

```
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 newStr = "";
      char ch ;
      for (int i = 0; i < s.length(); i++) {
      // check is char at index i is a letter
         if (Character.isLetter(s.charAt(i))) {
      // checks if char at index i is uppercase
            if ((s.charAt(i) >= 'A') && (s.charAt(i)<= 'Z')) {
               ch = Character.toLowerCase(s.charAt(i));
            } else {
               ch = s.charAt(i);
            }
         } else {
            ch = s.charAt(i);
         }

         newStr += ch;
      }
      return newStr;
   }
}
```

```
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 newStr = "";
        char ch;
        for (int i = 0; i < s.length(); i++) {
            ch = s.charAt(i);
            if (ch == ' ' || newStr.indexOf(ch) == -1) {
                newStr += ch;
            }}
        return newStr;
}
}
```

```
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 sunday = 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[]) {
      String YearInput = args[0];
      int yearInput = Integer.parseInt(YearInput);
      // 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 (dayOfMonth != 31 || month != 12 || year != (yearInput)) {
         advance();
         debugDaysCounter++;

         if (year == yearInput) {
            if (dayOfWeek == 1) {
               sunday++;
               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 (month == 12 && dayOfMonth == 31) {
        month = 1;
        dayOfMonth = 1;
        year++;
    }
    else if (dayOfMonth == nDaysInMonth) {
        month++;
        nDaysInMonth = nDaysInMonth(month, year);
        dayOfMonth = 1;
    }
    else {
        dayOfMonth++;
    }


}
public static int nDaysInMonth(int month, int year) {
    // Replace the following statement with your code
    int days = 31;
    switch (month) {
        case 2:
            if (isLeapYear(year)) {
                days = 29;
            } else {
                days = 28;
            }
            break;
        case 4:
            days = 30;
            break;
        case 6:
            days = 30;
            break;
        case 9:
            days = 30;
            break;
        case 11:
            days = 30;
            break;
    }
    return days;
}
// Returns true if the given year is a leap year, false otherwise.
public static boolean isLeapYear(int year) {
```

```
        // Replace the following statement with your code
        boolean isLeap = false ;
        isLeap = ((year % 400) == 0);
        isLeap = isLeap || (((year % 4) == 0) && ((year % 100) != 0));
        return isLeap;
    }
}
```