HW3Code Sapir Erlich -

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;    // 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 increment = 0.001;
     double g = loan/n;
     LoanCalc.iterationCounter = 0;
     while (endBalance(loan,rate,n,g) >= epsilon) {
        g += increment;
        LoanCalc.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) {
       // Sets L and H to initial values such that f(L) > 0, f(H) < 0,
       double h=loan;
       double l=0;
       double g = (l + h) / 2;
       LoanCalc.iterationCounter = 0;
       // implying that the function evaluates to zero somewhere between L and H.
       // So, let's assume that L and H were set to such initial values. //Setgto(L
+ H)/2
       while (h-l > epsilon){
           // Sets L and H for the next iteration
           if (endBalance(loan,rate,n,g) > 0){
               l=g;
           }
           // the solution must be between g and H // so set L or H accordingly
           else{
               h=g;
           }
           g = (l + h) / 2;
           LoanCalc.iterationCounter++;
           // the solution must be between L and g // so set L or H accordingly
           // Computes the mid-value (g) for the next iteration
       }
```

```
        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 endBalance=loan;
      for (int i=0; i<n;i++){
        endBalance=(endBalance-payment)*(1+rate/100);


      }
      return endBalance;
    }
}
```

2. LowerCase -

```java
/** 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 output="";
       for (int i=0; i < s.length();i++){
           if ((s.charAt(i) >= 'A') && (s.charAt(i) <= 'Z')){
              output=output+(char)(s.charAt(i)+32);
           }
           else {
               output=output+(char)(s.charAt(i));
           }

       }
       return output;
   }
}
```

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) {
       String output="";
       for (int i=0; i<s.length();i++){
           if (s.charAt(i)==' '){
               output=output+s.charAt(i);
           }
           else if (s.indexOf(s.charAt(i))==i)
           {
               output=output+s.charAt(i);
           }

       }
       return output;
   }
}
```

4. 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 calendar_year=Integer.parseInt(args[0]);
       //// Write the necessary initialization code, and replace the condition
       //// of the while loop with the necessary condition
       while (year <= calendar_year && month <= 12 && dayOfMonth <= 31 ) {
           if (year==calendar_year){
               System.out.print(dayOfMonth+"/"+month+"/"+year);
               if (dayOfWeek ==1){
                   System.out.print(" Sunday");
               }
               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() {
       Integer DaysInMonth=nDaysInMonth(month,year);
       if (dayOfMonth < DaysInMonth){
           dayOfMonth++;

       }
       else if (month == 12){
           year++;
           month = 1;
           dayOfMonth=1;

       }
       else{
           month++;
           dayOfMonth=1;
       }
       if (dayOfWeek<7){
           dayOfWeek++;
       }
       else{
           dayOfWeek=1;
       }

    }

   // Returns true if the given year is a leap year, false otherwise.
   private static boolean isLeapYear(int year) {
       if ((year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 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) {
       boolean isLeapYear= isLeapYear(year);
       if (month == 9 || month == 4 || month == 6 || month == 11){
           return 30;
```

```
        }
        else if (month == 2) {
            if (isLeapYear){
                return 29;
            }
            else {
                return 28;
            }
                    // Replace the following statement with your code

        }
        return 31;
    }
}
```