<u>Hw2 – neta tarshish</u>

LoanCalc.java

```
/**
* 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) {
        iterationCounter = 0;
  double g = loan / n;

  while (endBalance(loan, rate, n, g) > 0) {
    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 upperBoundPayment = loan;
        double lowerBoundPayment = 0;
        double currentTestedPayment = loan/2;
        while((upperBoundPayment - lowerBoundPayment ) > epsilon){
                   double balance = endBalance(loan,rate,n,currentTestedPayment);
                if( balance > 0 ){
                        lowerBoundPayment = currentTestedPayment;
                        currentTestedPayment =
(lowerBoundPayment+upperBoundPayment)/2;
                        iterationCounter+=1;
                }
                else if( balance < 0 ){
                        upperBoundPayment = currentTestedPayment;
                        currentTestedPayment =
(lowerBoundPayment+upperBoundPayment)/2;
                        iterationCounter+=1;
                }
          }
        return currentTestedPayment;
  }

        /**
        * 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) {
                for(int i = 0;i<n;i++){
```

```
            loan = (loan-payment)*(1+(rate/100));
        }
    return loan;

}
}
```

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) {
        char letter;
        String word = "";
        for(int i = 0; i < s.length();i++){
            if(s.charAt(i)>='A'&& s.charAt(i)<='Z'){
                letter = (char)(s.charAt(i)+32);
                word += letter;
            }
            else{
                word += s.charAt(i);
            }
        }
        return word;
    }
}
```

UniqueChars.java

```
/** 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 word = "" + s.charAt(0);
        Boolean check = false;
        for(int i = 1; i<s.length();i++){
            for(int j = 0;j<word.length();j++){
                if(s.charAt(i)==word.charAt(j)&&s.charAt(i)!=32){
                    check = true;
                }
            }
            if(!check){
                word += s.charAt(i);
            }
            check = false;
        }
        return word;
    }
}
```

Calendar.java

```
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar {
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;
    static int nDaysInMonth = 31;

    public static void main(String args[]) {
        int currentYear = Integer.parseInt(args[0]);
     int isSunday=firstDayOfYear(currentYear);
     System.out.println(isSunday);
     while(month<=12){
            while(dayOfMonth<=nDaysInMonth(month,currentYear)){
                if(isSunday%7==0){
                    System.out.println(dayOfMonth
+"/"+month+"/"+currentYear+" Sunday");
                }
                else{
                    System.out.println(dayOfMonth
+"/"+month+"/"+currentYear);
                }
                isSunday+=1;
                dayOfMonth+=1;



            }

        month+=1;
```

```
                   dayOfMonth = 1;


           }

        }

        public static int firstDayOfYear (int year){

                   int daysCounter = 1;

                   int startYear = 1990;

                   while(startYear<year){

                   for(int i = 1;i<13;i++){

                              daysCounter+=nDaysInMonth(i,startYear);

                   }

                   startYear+=1;

        }

        return (daysCounter%7);

        }




// Returns true if the given year is a leap year, false otherwise.

        private static boolean isLeapYear(int year) {

           int i = 0;

                   boolean check = false;

           while(2024 - (4*i)!=year&&(i*4)<=2024){

                   i+=1;

           }

           if(2024 - (4*i)==year){

                   check = true;

           }

                   return check;

                   }
```

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

            return 31;
    }
}
```