Tali Rafael – 209907633
HW03

```
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 oldString) {
      // Replace the following statement with your code
      String newString = "";

      for (int i = 0; i < oldString.length(); i++) {
         char currentChar = oldString.charAt(i);
         if (currentChar == ' ' || newString.indexOf(currentChar) == -1) {
            newString = newString + currentChar;
         }
      }
      return newString;
   }

}
```

Tali Rafael – 209907633
HW03

```
/** 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) {
      // Replace the following statement with your code
      String lowerCaseString = "";
         for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) >= 65 && s.charAt(i) <= 90) {
               lowerCaseString += (char)(s.charAt(i) + 32);
            }
         else {
                  lowerCaseString += s.charAt(i);
            }
         }
      return lowerCaseString;
   }
}
```

Tali Rafael – 209907633
HW03

```
/**
* 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(endBalance (100000, 5, 3, 10000));
		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) {
        // Replace the following statement with your code
            iterationCounter = 0;

            double L = loan / n;
            double H = loan;
            double g = ((H + L)/2);

            while ((H - L) > epsilon) {
                // Sets L and H for the next iteration
                if (endBalance(loan,rate,n,g)*endBalance(loan,rate,n,L) > 0) {
                // the solution must be between g and H
                // so set L or H accordingly
                L = g;
            }
                else {
                H = g;
            }
            g = (H + L)/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 endB = loan;
```

Tali Rafael – 209907633
HW03

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

                }
        return endB;

        }
}
```

Tali Rafael – 209907633
HW03

```
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 = 1;     // 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[]) {

   int finalYear = Integer.parseInt(args[0]);

      while (year < finalYear + 1) {
            if (year == finalYear) {
         String sundayPrinter = "";
         if (dayOfWeek % 7 == 0) {
            sundayPrinter = " Sunday";
         }

         System.out.println(dayOfMonth + "/" + month + "/" + year + sundayPrinter);
      }
            advance();
   }
      //// 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 void advance() {
         // Replace this comment with your code
         dayOfWeek++;
         dayOfMonth++;
         if (dayOfMonth > nDaysInMonth) {
               dayOfMonth = 1;
               month++;
               if (month > 12) {
                     year++;
                     month = 1;
               }
         }
```

```
            nDaysInMonth = nDaysInMonth(month, year);
        }

    // 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) {
            if (month == 4 || month == 6 || month == 9 || month == 11) {
                return 30;
            }
            else if (month == 2 && isLeapYear(year)){
                return 29;
            }
            else if (month == 2 && !isLeapYear(year)) {
                return 28;
            }
            else {
                return 31;
            }
    }
}
```