CSIS 10B **Assignment 2** **SOLUTIONS**

Problem1 CodingBat

```
/**
 * CodingBat problems.
 *
 * @author Daniel Vazquez
 * @version 02272017
 */
public class CodingBat
{   // copy all your solutions from coding bat problems here

    /////////////// countEvens ///////////////
    public int countEvens(int[] nums) {
        int numEven = 0;
        for(int k = 0; k < nums.length; k++){
            if(nums[k]%2 == 0){
                numEven++;
            }
        }
        return numEven;
    }

    /////////////// only14 ///////////////
    public boolean only14(int[] nums) {
        for(int k=0; k < nums.length; k++){
            if(nums[k] != 1 && nums[k] != 4){
                return false;
            }
        }
        return true;
    }

    /////////////// shiftLeft ///////////////
    public int[] shiftLeft(int[] nums) {
        if(nums.length>1){
            int first = nums[0];
            for(int k = 1; k < nums.length; k++){
                nums[k-1] = nums[k];
            }
            nums[nums.length-1] = first;
        }
        return nums;
    }

    /////////////// sum13 ///////////////
    public int sum13(int[] nums) {
        int sum = 0;
        if(nums.length > 0){
            for(int k=0; k < nums.length; k++){
                if(nums[k]!=13){
                    sum += nums[k];
                }else{
                    k++; // if 13 is found, skip next number, too.
                }
            }
        }
        return sum;
    }

}
```

Problem2.java

```
import java.util.Scanner;
/**
 *  a Java program that uses an ArrayStack (a Stack based on an Array)
 *  and a LinkedQueue (both holding Character data) to test whether an
 *  input string is a palindrome.
 *
 * @author Daniel Vazquez
 * @version 02272017
 */
public class Problem2
{
    public static void main(String [] args)
    {
        // Solve the Palindrome problem here
        System.out.println("--------- Palindrome checker ---------");

        // variables to use
        String palindrome; // holds palindrome
        boolean isPalindrome = true;
        // Last-In-First-Out (LIFO) policy.
        ArrayStack<Character> stack = new ArrayStack<Character>();
        //  First-In-First-Out (FIFO) policy.
        LinkedQueue<Character> queue = new LinkedQueue<Character>();

        // scanner object
        Scanner keyboard = new Scanner(System.in);
        // read line of text from keyboard

        // prompt user to input text
        System.out.print("Enter line of text: ");

        // read line of text
        palindrome = keyboard.nextLine();

        // convert string to lowercase
        palindrome = palindrome.toLowerCase();

        // show entered text
        System.out.println("Entered text: " + palindrome);

        // loop to insert letters of string to both the stack and queue
        for(int k=0; k < palindrome.length(); k++){
            // check if character at position k is a letter
            if(Character.isLetter(palindrome.charAt(k))){
                // if character is a letter, add it to both, the stack and queue
                stack.push(palindrome.charAt(k)); // add letter to the top
                queue.enqueue(palindrome.charAt(k)); // add letter to the tail
            }
        }
        // A Stack has a Last-In-First-Out policy and a
        // Queue has a First-In-First-Out policy
        // we can determine if a text is a palindrome if we
        // compare the first and last character,
        // and continuing comparing the text's ends in an
        // inward direction. By removing a character from
        // the stack (the last character added) and a charecter
        // from the queue (first one added) we are able to
        // compare letter by letter, from the
        // ends to the center of the string.
        while(!stack.isEmpty()&&!queue.isEmpty()){
            if(stack.pop()!= queue.dequeue()){
                isPalindrome = false;
                break;
            }
        }

        System.out.print("The text \"" + palindrome + "\" "
                    + (isPalindrome?"is":"is not")+ " a palindrome!");

    }
}
```

Problem3.java

```
import java.util.Scanner;
import java.io.*;
/**
 * This program reads and processes a list of students
 * It will go through and remove all of the Students who fall below
 * a given GPA cutoff, and will print the resulting list of remaining
 * students.
 *
 * @author Daniel Vazquez
 * @version 03072017
 */
public class Problem3
{
    public static void main(String [] args)
    {
        // solve the Student List problem here
        // variables to use
        File file; // file object
        Scanner keyboard; // keyboard input
        Scanner fileInput; // file input
        double gpaCutOff; // GPA cutoff
        int studentsDel = 0; // numbers of students deleted
        LList<Student> students = new LList<Student>(); // list of students

        // create a File object initialized to Student100.txt
        file = new File("Student100.txt");

        // try to open and retreive data from text file
        try{
            // initialize file's scanner object
            fileInput = new Scanner(file);
            // read a new line as long as it is available
            while(fileInput.hasNext()){
                // "extract" a student from the file text
                Student tempStudent = new Student(fileInput);
                // add student to the student's list
                students.add(tempStudent);
            }
        }catch(IOException e){
            System.out.println("Input error! " + e.getMessage());
        }catch(Exception e){
            System.out.println("Something went wrong! " + e.getMessage());
        }

        // sort students by first letter of last name
        SortLastName(students);

        // print the list
        System.out.println("-------- All students --------");
        for(int k=1; k <= students.getLength(); k++)
            System.out.println(students.getEntry(k).toString());


        // create a scanner object to read from keyboard
        keyboard = new Scanner(System.in);

        // ask the user to input a cutoff GPA value
        System.out.print("\nEnter GPA cutoff: ");

        gpaCutOff = keyboard.nextDouble();

        // loop to go through and remove all of the Students who fall
        // below the user's given GPA cutoff
        // Using students.getLength() will return the updated size
        // so we don't have to worry about nullpointerexception
        int position = 1; // student position
        while(position <= students.getLength()){
            if(students.getEntry(position).getGPA() < gpaCutOff){
                System.out.println("Deleted: " + students.getEntry(position).toString());
                students.remove(position); // delete student a position k
                studentsDel++;  // increment student counter
                // For LList (linked list), everytime a object is deleted from
                // a position x, the following objects are "shifted" so they
                // occupy the empty space left by the deletion.
                // In other words, immnediately after deleting an object at
                // position x, the next object takes its place. For that reason
                // we do not need to increase the position
            }else{
                // increase the "index" by one
                position++;
            }
        }

        /// ------------- ALTERNATIVE: USING FOR LOOP -----------
        /*
        for(int k=1; k <= students.getLength(); k++){
            if(students.getEntry(k).getGPA() < gpaCutOff){
                System.out.println("Deleted: " + students.getEntry(k).toString());
                students.remove(k); // delete student a position k
                studentsDel++; // increment student counter
                k--; // since "students" is a LList (linked list), everytime a
```

```
                     // student is removed, all the students are "shifted" to
                     // occupy the space left by the removed student.
                     // The shift is, in reality, a pointer manipulation inside
                     // the class LList.
                     // For that reason if using a "for" loop we need to check
                     // the current position again since it will contain a new
                     // student. we subtract -1 to the current position so
                     // the next loop iteration (k++) will result in the
                     // current position
            }
        }*/

        // show number of students deleted
        System.out.println("Students deleted: " + studentsDel);

        // show list of students whose GPA > gpaCutOff (given GPA Cutoff)
        System.out.println("\n---- Students with GPA >= "+ gpaCutOff +" ----");
        for(int k=1; k <= students.getLength(); k++)
            System.out.println(students.getEntry(k).toString());

    }

    /**
     * Sorts the given LList<Student> by alphabetical order of first letter
     * of last names.
     * It removes all students whose last name begins with ‘A’, and inserts
     * them at the end of the list. It repeats the process with students whose
     * last name begins with ‘B’, ‘C’, ‘D’, etc. At the end of this process,
     * all the A-name students will appear at the beginning, then B-name next,
     * and so on, with Z-name students at the end.
     *
     * @param  students the list of students to be sorted
     */
    public static void SortLastName(LList<Student> students){
        // String to store the last name initials in the order in which
        // they should be proccesed, in this case: alphabetical order
        String targetLetter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        //int listSize = students.getLength();
        // Outer loop: change the target letter
        for(int k = 0; k < targetLetter.length(); k++){
            // since students will be moved to the end of the list
            // we must avoid compare them again. compareUpTo will
            // represent the remaining number of users that need
            // to be compared
            int compareUpTo = students.getLength();
            // Inner loop: process the list of student
            // note that loop is missing the counter. The reason is that
            // every time we move a student to the end of the list
            // all students are shifted to the front. if the counter
            // increments after moving a student, then the loop will
            // skip that position which is occupied by a new student.
            for(int i = 1; i <= compareUpTo; ){
                // check if current student's lastname begins with targetLetter
                if(students.getEntry(i).getLastName().charAt(0) ==
                   targetLetter.charAt(k))
                {
                    // copy current student to the end of the list
                    students.add(students.remove(i));
                    // reduce limit
                    compareUpTo--;
                }else{
                    // increment the counter only if no student was moved
                    // to the end of the list
                    i++;
                }
            }
        }
    }
}
```