Regis University CC&IS
CS310 Data Structures
Programming Assignment 3: Linked Lists

Problem Scenario

The charity was very impressed with your work from last week. Nevertheless, the IT director at the charity just read his "Java Geek Weekly", and discovered ArrayLists may not be the most optimal data structure to use. Thus, the charity has asked to replace the ArrayList you developed last week with Linked Lists.

There was also some concern within the charity there were too many errors being allowed into the reports. They would still like the program to first run the report as you did last week, producing a report in the file:
`output/assn3initialReport.txt`

But after that, they would like the program to validate the data in the linked lists, and clean them up. The charity would like the program to:

- Check the Donor list for invalid email addresses, using the email validation method already developed for the Donor class
  - If an address is invalid, remove the donor from the donor list, along with all of the donor's donations in the donation list.
- Check the Donation list for invalid check numbers, using the check number validation method already developed for the Donation class
  - If a donation has an invalid check number, remove that donation from the list.
- Write lines to the console, as it cleans the lists, indicating any actions taken to remove invalid data.

Once all the data has been validated, the program should run the final report again, producing a second report in the file: `output/assn3cleanReport.txt`
so the charity can compare the two reports.

Program Requirements

The program must follow the **CS310 Coding Standards** from Content section 1.9.

This **csv** input file for this program will be called "assn3input.txt", and will have the same format as the input file for Assignment 2.

The **Implementations** from last week will be ***replaced*** with new implementation classes that will implement Linked Lists. So you will be modifying the implementations from last week, by removing the ArrayLists and replacing them with this week's Linked List structures.

The textbook described several types of linked lists, including:

- Singly Linked Lists
- Java Collection Linked List (java.util.LinkedList)

Your **DonorLogImpl** will implement an ***ordered*** singly linked list, with a node class and methods you code.

Your **DonationLogImpl** will implement a Java Collection Linked List.

And to enable writing to different files, you need to add a filename parameter to the method that invokes **PrintImpl**'s report printing method, and to the **PrintImpl**'s report printing method itself.

Iteration

The **DonorLogImpl** methods will traverse the singly linked list, starting at the top, using a ***for*** or ***while*** loop.

> Since this is an ordered list, when adding new nodes, the code will need to find the correct insertion spot that will maintain order by donor ids.

The **DonationLogImpl** methods will use **Iterator**, as described in the textbook, to traverse the linked list.

New Methods

Add a **traverseDisplay()** method to both **DonorLogImpl** and **DonationLogImpl**.
These methods will first display a header:

`Donor List:`   OR   `Donation List:`

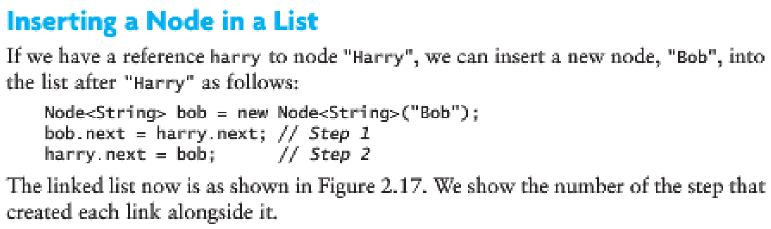
and will traverse the list being implemented, using the **toString()** method to display each object in the list.

Add a **cleanUp()** method to the Donor implementation to validate and clean up the donor list.

Add a **cleanUp()** method to the Donation implementation to validate and clean up the donation list.

Node Class

The textbook refers to inner classes – you have probably not seen these yet. Inner classes allow you to include one class within another. Using the inner class, the textbook examples accessed attributes like this



Inserting a Node in a List

If we have a reference harry to node "Harry", we can insert a new node, "Bob", into the list after "Harry" as follows:

```
Node<String> bob = new Node<String>("Bob");
bob.next = harry.next; // Step 1
harry.next = bob;      // Step 2
```

The linked list now is as shown in Figure 2.17. We show the number of the step that created each link alongside it.

Since all of the attributes were private, and within the parent class, it was possible to use the "dot" (.) notation to access the attributes.

You will ***not*** do this. Instead of using inner classes, you will be creating your own **Node** class for the Donor's singly linked list. Since the **Node** class will be a separate class, you will use getter/setter methods for access to private attributes. For example, using a **Node** class, the above code would be:

```
Node<String> bob = new Node<String>("Bob");
bob.setNext(harry.getNext());      // step 1
harry.setNext(bob);                // step 2
```

A sample **Node** class for a singly linked list is shown in the linked list building example of the online Content in section 3.2.

The main method

You will start with the same **main** method that you used for Assn 2.
After reading the data and creating the initial report, add the code to the end of the **main** method to:

- Display each list, using the **traverseDisplay()** methods.
- Clean up the lists.
- Create the second report.

Hints

Your application should contain the following class source code files now:

- CS310<lastname>.java
- Node class for a singly linked list
- DonorListImpl – singly linked list of your implementation
- DonationListImpl –using **java.util.LinkedList**
- PrintImpl – same format as last week, but modified to use the linked lists

Deliverables

- Your input data file will still be read from the **input** folder in your project.

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:
  `assn3input1.txt`
  `assn3input2.txt`
  (i.e. number each data file after the filename of **`assn3input.txt`**)

  Together, all of your test data files should demonstrate that you have tested every possible execution path within your code, including erroneous data which causes errors or exceptions.

- Your output data files will still be written to the **output** folder in your project.
- Create and/or modify **Javadoc headers**, and generate **Javadoc files**
- Add screen shots of **clean compile** of your classes to the documentation folder.

WARNING: Submittals without the clean compile screenshots will not be accepted.

Program Submission

This assignment is due by midnight of the date listed on the **Course Assignments by Week** page.

- Export your project from NetBeans, following the same steps used in Assns 1 &2.
  - Name your export file in the following format:
    `CS310<lastname>Assn<x>.zip`

    For example:
    `CS310SmithAssn3.zip`

    NOTE: Save this zip file to some other directory, not your project directory.
- Submit your **.zip** file to the **Prog Assn 3** dropbox.

  Warning: Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

This program will be graded using the **rubric** that is linked under **Student Resources** page.

WARNING:
Programs submitted more than 5 days past the due date will ***not*** *be accepted,*
and will receive a grade of 0.