Course: CSC 115 (J)

Lab 05: Linked List

Preparation

Make sure you understand Parts 1 & 2 of Lab 4, including the inheritance structure and the array-based implementation of the *EnrollmentList* interface that you completed in *Course.java*

Objectives

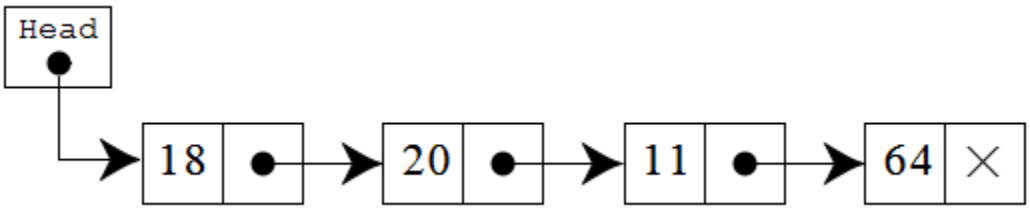
You will be using some of the code from Lab 4 to implement a linked list that stores people associated with a course, including students, and TAs.

During lab 5, you will:

- Write a Node class that stores a *Person* object as its data.
- Implement the *EnrollmentList* interface from Lab 4 using a linked list data structure.
- Come to better understand adding and removing from a linked list.

Background

A linked list is a data structure consisting of a group of nodes where each node is composed of a datum and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence.



A linked list whose nodes contain two fields: an integer value and a link to the next node. A head pointer is pointing to the first node and the last node has null value in its link field.

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including *stacks*, *queues*, *trees*, and symbolic expressions. It is not uncommon to implement the other data structures directly without using a list as the basis of implementation.

For more details, visit http://en.wikipedia.org/wiki/Linked_list

Part I - The Node Class

1. Download *Person.java*, *Student.java*, and *TA.java* from connex.
2. Download and complete *Node.java* so that it includes the following features:
   - Each *Node* should store a *Person* object, in addition to a reference to the next *Node* in the linked list.
   - A constructor that takes in a *Person* as a parameter.
   - *Getters* and *setter* methods for its instance variables.
   - A *toString()* method that returns the data being stored in the *Node* as a String.
3. Make sure your Node class compiles before moving on to the next step.

Part II - Linked List

Download *CourseLL.java* and *EnrollmentList.java* from connex. *EnrollmentList* is an interface that contains a list of functions that your program will need to include. This part will have you complete *CourseLL.java*, such that it implements all the functionality specified in the *EnrollmentList* interface using a linked list data structure, as opposed to the array-based program you worked on in Lab 4.

Complete the *CourseLL* class so that it includes the following features:

- A *CourseLL* should have a name, an integer that stores the number of people in the course, and a reference to the head of the linked list.
- A constructor with the following signature:
  - `public CourseLL(String courseName)`
- A *toString()* method that returns the course's information as a single *String* with the following format:

```
CSC 115 Class List
TA: Ackbar, admiral@rebels.com, Employee#: 4325432, Salary: 10000, Student#:
432423
Student: Ackbar, admiral@rebels.com, Student#: 54354353, Grade: 94
```

- The *CourseLL* class should **implement** the *EnrollmentList* interface. This means that it **must** include all the methods described in the interface; otherwise you will get compiler errors.
- Fill in the methods one by one, making sure to test each one before you move on to the next. Use the comments provided with the interface to figure out what each method is supposed to do.
- The **remove** method is the most challenging, and should follow the approach given below:

- If the head of the list contains the item to be removed, remove the first node in the list.
- Otherwise, loop through the linked list while keeping track of the current node and the node right before it (the previous node), so that you can remove the current node if it is warranted by changing the links of the previous node.

Part III - Linked List with a Tail

The linked list implementation from part two works efficiently so long as we are content with only adding to the front of the list. If we want to be able to add quickly to either end of the linked list, we need to modify *CourseLL.java* so it tracks not only the **head** of the linked list, but also the end of the linked list (called the **tail**).

Change CourseLL.java so that it tracks the tail of the linked list, and adds new items at the end of the linked list.

- Add the tail as an instance variable.
- Changes must be made to the add method so that it adds new items to the end of the list, instead of at the beginning of it.
- The remove method must be changed to ensure that the tail is always pointing at the last node in the list.

Have fun