# Overview

In this lab, you will create methods to add functionality to a LinkedList. You will be implementing several different methods that require you to traverse (and build) lists. You will be working with a simplified version of a linked list class that is built from nodes that are public. The nodes have a pointer to the next node and two other fields, a String named key and an int named value. The list is a singly linked list. You should look at the details of the LinkedList class first to get an idea of how to access its functionality.

# Objectives

* Practice working with a class
* Creating and using non-static methods
* Practice with linked lists
* Applying test cases to your program

# Steps

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Download the starter code from the course public folder: [public/10L](https://cs.unh.edu/~cs416/public/10L). Put these files into the src folder. You will be adding methods to LinkedList.java.
5. Now it’s time to start coding:
   1. Create the following inside LinkedList.java:
      1. A non-static void method called **unlinkNode** that takes a Node n as a parameter. It should ensure that the node is unlinked from the nodes before and after it. You will need to change the prev of the node after n and the next of the node before n. Remember that if n is the head of this list, it will have no node before it (but the value of head will need to move past it) and if n is the tail of this list, it will have no node after it (but the value of tail will need to move before it.)
      2. A non-static void method called **addAfter** that takes a Node n and a Node before as parameters. This method inserts n to be in the list just after the node named before. You will need to change the next from before, the prev from the node following before, and assign n's next and prev.
      3. A non-static method called **swapIfNeeded** that returns a boolean and takes a Node n as a parameter. It compares the key of n with the key of the node after n, swapping the two if n's key is greater than the next node's key. Remember that if n is the tail of this list there is no node after, so it does not ever need to be swapped. It returns true if the nodes needed to be swapped, and false otherwise.  
           
         One easy way to swap the nodes would be to unlink n from the list and then add it after the other node (which you would have to store in a temporary variable.)
      4. A non-static void method called **sort** that takes no parameters. Its goal is to sort the list in ascending order by key. It will use an outer loop that continues until it is sure the list is sorted (for which you will need a boolean variable.) Inside this loop it then iterates through this list (starting at the head) and swaps any nodes that are out of order. If any swaps were needed, it records that the list is not sorted and continues the outer loop. It can stop the outer loop when an iteration through the list finishes without any swaps.