# Linked List Lab 

The aim of this lab is to implement a LinkedList object that is similar in mechanics and efficiency as the one in `java.util.LinkedList`. You will find folders for `Linked List LAB`, `Circular Linked List LAB`, and `Double Linked List LAB`. Only one lab needs to be submitted at the end, but it is recommended that you begin with `Linked List LAB`. Once completed, it will be easy to transition to a more sophisticated one that will yield a higher grade.

Within each folder, you will find a class file for the data structure, an interface the container needs to implement, a node class to define each cell of the container and a driver program to test the code:

In the folder `Linked List LAB`:

- `LinkedList.java`, where you will code the data-fields and logic for the data-structure.
- `ListInterface.java`, which defines the interface that LinkedList will implement.
- `ListNode.java`, defines a single cell of the container. A LinkedList will chain many nodes.
- `LLDriver.java`, which has a main function to test your data-structure.

The grade you receive will depend on the efficiency of the most advanced version of a Linked List that you submit: the better the average efficiency, the higher the grade. Upon completion of the regular LinkedList, you will have time to evolve your data structure to one that is more efficient until you get to the grade that you want.

Here is the rubric for grading the final version of a Linked List that you submit:

- 85% if the following features all have O(n) efficiencies (they require loops to traverse): a method that returns the size, adding a new last element, remove the last element, set the last element to a new value , get the last element.

- 90% if the size method is O(1), meaning that it does not contain a loop that traverses through the list, but still O(n) to add a new last element, remove the last element, set the last element, get the last element.

- 95% if size, add a new last element, set the last element and get the last element are all O(1), but remove the last element is O(n).

- 100% if size, add last, remove last, set last and get last are all O(1), meaning that none of these abilities require a loop to traverse. This will require either a Double Linked List with a tail-pointer, or a list that is both double-linked and circular.

Douglas R Oberle, 2024