COSC 121: Computer Programming II

Today's Key Concepts

- Implementing data structures from scratch
  - An ArrayList using an array
  - A LinkedList using a Node structure
- Handling nuances
- Developing your own test cases

Recall: ArrayList (Dynamically Growing Array)

boolean add(E o)	Appends a new element o at the end of this list.
void add(int index, E o)	Adds a new element o at the specified index in this list.
E set(int index, E e)	Sets the element at the specified index.
E get(int index)	Returns the element from this list at the specified index.
boolean remove(Object e)	Removes the first occurrence of element o from this list.
E remove(int index)	Removes the element at the specified index.
void clear()	Removes all the elements from this list.
int indexOf(Object e)	Returns the index of the first matching element in this list
int lastIndexOf(Object e)	Returns the index of the last matching element in this list.
boolean contains(Object e)	Returns true if this list contains the element o.
boolean isEmpty()	Returns true if this list contains no elements.
int size()	Returns the number of elements in this list.
void trimToSize()	Trims the capacity of this ArrayList instance to the current size.
boolean equals(Object x)	Returns true if **x** is a list and both lists have the same size, and all corresponding pairs of elements are *equal*.
String toString()	Returns a string representation of the list elements.

MyList.java

Implement your ArrayList using DynamicArray.java

Exercise: Implement ArrayList Methods

- What additional methods do you need for good practice? Consider test cases:
  - Positive test case: When you expect the method to work
  - Negative test case: When you expect the method to not work/fail
  - Boundary test case: Edge cases that work but may be handled differently

Examples:

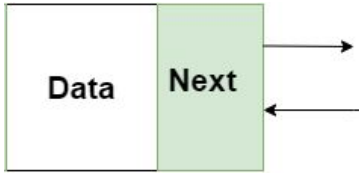
- What happens if array is full but you need to add?
- What if the index given for the setter is out of bounds?
- Retrieving last instance of an element that occurs only once?

Exercise: Implement ArrayList Methods

- What additional methods do you need for good practice? Consider test cases:
  - Positive test case: When you expect the method to work
  - Negative test case: When you expect the method to not work/fail
  - Boundary test case: Edge cases that work but may be handled differently
- Try implementing:
  - **public void** add(E elem)
  - **public** E set(**int** index, E elem)
  - **public** E get(**int** index)
  - **public boolean** remove(E elem)
  - **public int** indexOf(E elem)
  - **public boolean** contains(E elem)
  - **public int** size()

Implementing Linked List

- List now contains nodes
- Singly linked list:




```
public class Node<E> {
    E element;
    Node<E> next;
    public Node( E elem ) {
        element = elem;
        next    = null;
    }
}
```

- Doubly linked list:


```
public class Node<E> {
    Node<E> prev;
    E element;
    Node<E> next;
    public Node( E elem ) {
        prev    = null;
        element = elem;
        next    = null;
    }
}
```

Review linked list operations and check out MyLinkedList.java