Objective

- Learn how to implement a LinkedList
- Learn how to use a private class
- Practice using Generics
- Practice calculating Big O

Setup

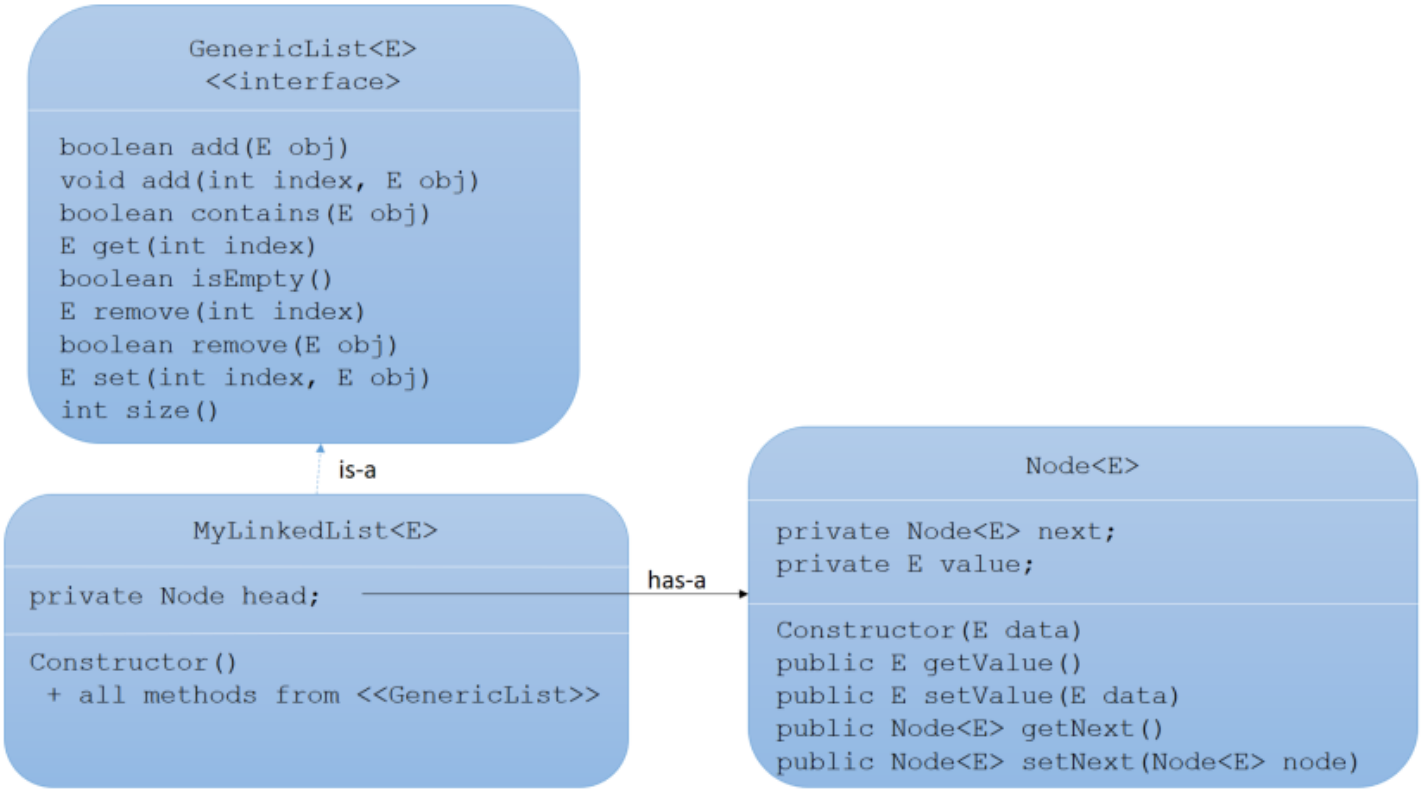
1. Download the `Lab12StarterCode.zip` file from `msBrekke.com`
2. Unzip starter code in `H:/CSIII/Lab12` folder

Overview

You are going to implement a LinkedList data structure.

Implementing a LinkedList

In `MyLinkedList.java` write the `MyLinkedList` class that implements the `GenericList` interface. You will also implement a `private Node` class inside the `MyLinkedList.java` file.



Let's Begin! (START CODING HERE):

NOTE: *None of your code will compile successfully at the beginning.* Follow the instructions below to get started.

1. Implement the `GenericList` interface. Note: when implementing a Generic interface you must include the angle brackets like this:

```
public class MyLinkedList<E> implements GenericList<E>
```

At the bottom (or end) of the `MyLinkedList` class, create a private `Node` class. This private class must be INSIDE your `MyLinkedList` class. `MyLinkedList` will have a structure as follows:

```
public class MyLinkedList<E> implements GenericList<E>
{

      //Private Helper Class
      class Node<E>
      {

      }
}
```

2. Inside the Private Helper Class, add two private instance variables: `value` and `next`.
`value` is a type `E` and `next` is type `Node<E>`.

3. The *constructor* will have a parameter of type type `E` that sets the instance variable `value`.

4. Write the getter and setter methods:

```
public E getValue()
{

}

public E setValue(E data)
{

}

public Node<E> getNext()
{

}

public Node<E> setNext(Node<E> node)
{

}
```

Now you are finished with the private helper class! You will now implement everything needed for `MyLinkedList` : the constructor, instance variables, and all methods defined in the `GenericList` interface.

5. Create a private instance variable for the `MyLinkedList` class: `head`.
   `head` is type `Node<E>`.

6. The default constructor for needs to create a `head` Node object which initially points to `null`.

```
public MyLinkedList()
{
    head = new Node<E>(null);
}
```

7. Add the methods defined in the `GenericList` interface. **See step 8 for advice!**

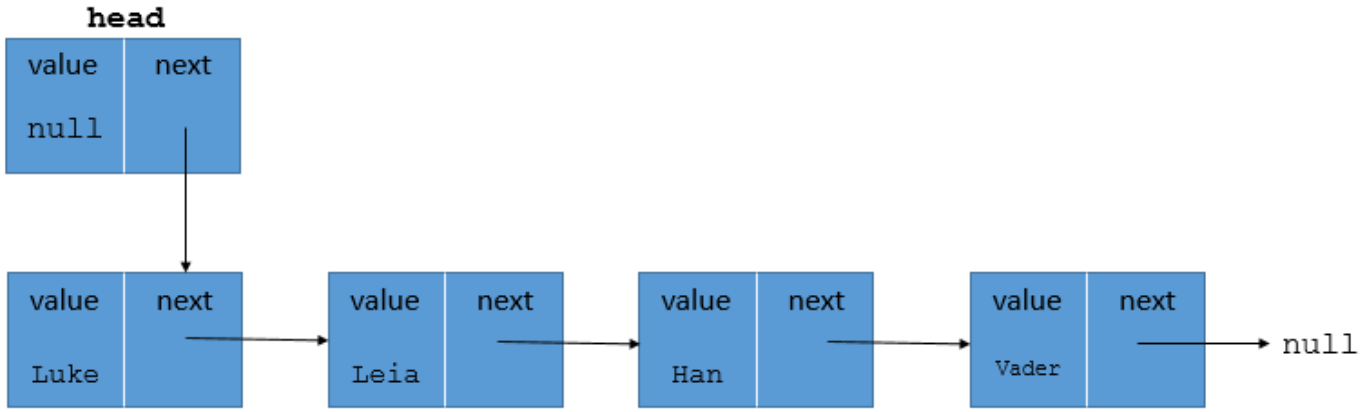
8. In your `MyLinkedList` class you should write two private methods:

```
/**
 *  This method returns the index where the first occurrence of obj
 *  If list does not contain obj, then return -1
 */
private int indexOf(E obj)

/**
 *  return the Node object that is stored at index
 *  If index is out of bounds then return null
 */
private Node<E> getNode(int index)
```

When trying to write your LinkedList methods, try to think of how you can put the algorithm in terms of `getIndex(obj)` and `getNode(index)`.

9. Remember that a *LinkedList* stores data in `Node` objects that each have a reference to the *next* `Node` in the list.



The instance variable `head` is a special `Node` object that **does not contain a value** and **always points to the first `Node` in the list**. The constructor should initialize this instance variable like below. (This was done in step 5.)

```
head = new Node<E>(null);
```

If the list is empty then `head.getNext()` should return `null` but the value of `head` should never be `null` itself.

When you are searching for a particular value/index in a LinkedList you need to loop through each `Node` in the list. You can often achieve this goal with a `while` loop like this:

```
//Create a temporary Node variable to store the node you are looking at
//It starts by pointing to the first node in the list
Node n = head.getNext();

//Check if you have reached the end of the list, or
//if you have found the Node that contains the value you are looking for
while( n != null && !n.getValue().equals(valueYouAreSearchingFor) )
{
  //if you haven't found the node you are looking for, then
  //move on to the next Node in the list!
  n = n.getNext();
}

//Now, the value of n is either null or it is a Node that contains
//the value you are looking for!
```

Note: this loop will look slightly different if you are searching for the `Node` at the k^{th} index. The differences are left as an exercise to the reader.

10. When you are <u>adding</u> a node to the list, you will use the looping method described above.

```
//Create a temporary Node variable to store the node you are looking at
//It starts by pointing to the first node in the list
Node n = head.getNext();

//Check if you have reached the end of the list, or
//if you have found the Node that contains the value you are looking for
while( n.getNext() != null )
{
      //set n to be n's next node
}

//When you leave the loop, setNext to be a new Node with the passed obj
n.setNext(new Node<E>(obj));

//return
```

You can use these *helper* methods to simplify the public methods you want to write. For example, you can remove the value at index `k` with the following algorithm:

1. getNode from index `k-1`, call this `last`
2. get the node that comes *after* `last`, call this `old`
3. get the node that comes *after* `old`, call this `next`
4. set the `nextNode` of `last` to `next`
5. return the value stored in `old`

Your `MyLinkedList` class should get all Green boxes in using the `Lab12Tester` test file.

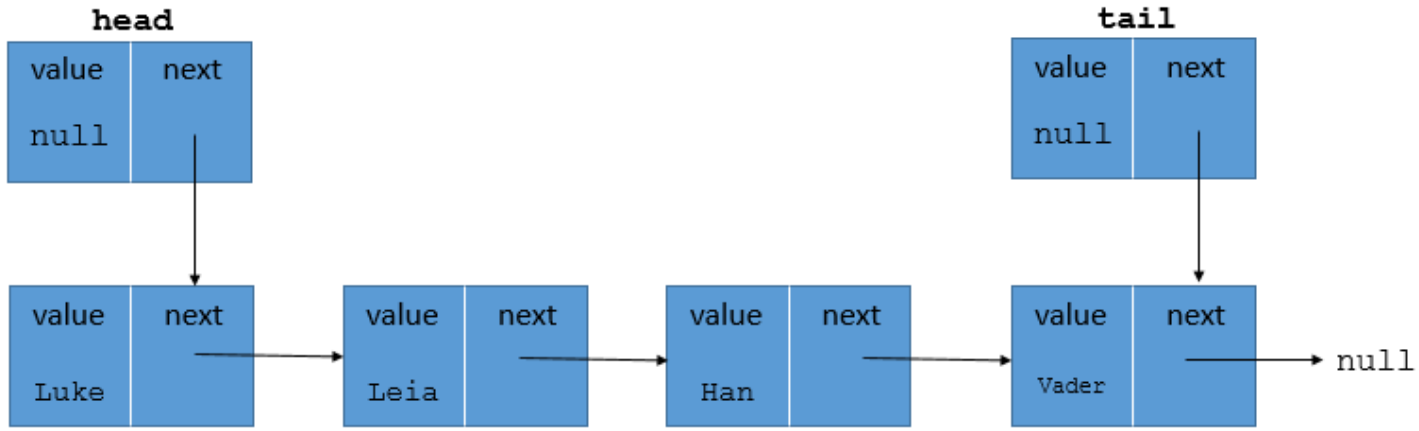
Questions

1. Identify the Big O of each of the methods you implemented in `MyLinkedList`

Method	Big O
`add(E obj)`	O(??)
`add(int index, E obj)`	O(??)
`contains(E obj)`	O(??)
`get(int index)`	O(??)
`isEmpty()`	O(??)
`remove(int index)`	O(??)
`set(int index, E obj)`	O(??)
`size()`	O(??)

2. Compare the Big O of `MyLinkedList` to the Big O of `ArrayList`
   a. Which methods are more efficient in `MyLinkedList`
   b. Which methods are more efficient in `ArrayList`?
   c. Which methods are the same efficiency in `MyLinkedList` and `ArrayList`?
   d. Describe a situation where it would be better to use `MyLinkedList` rather than `ArrayList`
   e. Describe a situation where it would be better to use `ArrayList` rather than `MyLinkedList`

3. Imagine if `MyLinkedList` had a second `private` instance variable called `tail`. Like the `head` instance variable, `tail` is a `Node` that always points to the *last* `Node` in the list.



   a. How would this additional instance variable improve the efficiency of the `add(E obj)` method?
   b. Would the `tail` instance variable improve the efficiency of any other methods? If so, which ones and how?

4. Imagine if `MyLinkedList` had another private instance variable called `size`. This instance variable is an `int` and it keeps track of how many elements are currently in the list. The `size()` method could be rewritten to:

```
public int size()
{
  return size;
}
```

   a. How would this change the Big O of the `size()` method?
   b. List all the methods that would have to be updated to either `++` or `--` the value of the `size` instance variable

Grading

Your program should meet all of the following specifications:

- Your code is in your `H:/CSIII/Lab12` folder
- Your code compiles and runs without error
- You only wrote code in the `MyLinkedList.java` and `LinkedListTester.java` files
- All Lab12Tester tests pass
- All questions are answered and turned in