Computer Science II
Handout 11

Linked Lists – master list of methods (so far)

1. `void addToFront()`
2. `boolean isEmpty()`
3. `LinkedList()`
4. `void clear()`
5. `String getFrontData()`
6. `Node getFrontNode()`
7. `String toString()`
8. `int size()`
9. `void removeFront()`
10. `void addToEnd(String d)`
11. `void removeLast()`
12. `int contains(String d)`
13. `void add(int index, String d)`
14. `void remove(int index)`
15. `Node getNode(int index)`

Linked Lists – operating on multiple lists

To work with more than one LinkedList, we will need the *front* node from each

Write methods for the following:

1. Add all elements from one LinkedList to the other
2. Merge the elements from two LinkedLists to create a third LinkedList
3. Extract a portion (sub-list) of a LinkedList, based on index

Multiple Linked Lists – Method 1

1. Add all elements from one LinkedList to the other

How do we traverse the *other* LinkedList?

```
public void addAll(LinkedList other) {



}
```

Multiple Linked Lists – Method 2

2. Merge the elements from two LinkedLists to create a third LinkedList

This makes more sense as a static method.

```
public static LinkedList merge(LinkedList first, LinkedList second) {
        LinkedList result = new LinkedList();









        return result;
}
```

Multiple Linked Lists – Method 3

3. Extract a portion (sub-list) of a LinkedList, based on index

What are valid input values for **`start`** and **`end`**?

```
public LinkedList subList(int start, int end) {
        LinkedList result = new LinkedList();
        if(!(start >= end ||                                    )) {









        }
        return result;
}
```

LinkedLists – operating on multiple lists

Now extend class LinkedList to class Set, which represents a list with non-duplicate elements.

Write methods for the following:

1. Union: the total of all Set elements, removing duplicates
2. Intersection: the elements that are shared between both Sets

```
public class Set extends LinkedList
```

Sets – Method 1

1. Union: the total of all Set elements, removing duplicates

```
public static LinkedList union(LinkedList first, LinkedList second) {
        LinkedList result = new LinkedList();









        return result;
}
```

Sets – Method 2

2. Intersection: the elements that are shared between both Sets

```
public static LinkedList intersection(LinkedList first, LinkedList second) {
      LinkedList result = new LinkedList();









      return result;
}
```

Other varieties of Linked Lists

- *Doubly Linked Lists* add an additional attribute to each Node: **`previous`**
  - This stores a reference to the previous Node in the list

- These are useful when reversing through the list will be a common operation
  - Most LinkedList methods should require only small changes: instead of only updating **`next`** within each Node, **`previous`** must also be accounted for

- *Circular Linked Lists* are a chain of nodes, where the last node has the first node as its **`next`** pointer
  - Could be used in sharing resources, like round-robin processor scheduling