List ADT & Linked Lists
The List Abstract Data Type

- List is an “ordered” sequence of data known as elements
- Ordered: each element has a position in the list (like in an array from 0 to the length of the list -1)
  - Ordered in this context does not mean that the list elements are sorted by value
- A list is said to be empty when it contains no elements
- The number of elements currently stored is called the length of the list
- The beginning of the list is called the head, the end of the list is called the tail
What Does a List Do?

- We’ll see the full Interface soon, but here’s a preview.
- Arrays are useful for storing ordered sequences of elements, but they have one significant drawback:
  - Can’t change the size of an array object once it’s created!
- A List can help us improve on the concept of an Array
  - Should still have some way of getting/setting the element at a particular index
  - Should also have the option of adding a new element anywhere in the List
List of ints

8 9 17 39 1092 3 4 -9
List of ints

Lists are **ordered**.

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>17</th>
<th>39</th>
<th>1092</th>
<th>3</th>
<th>4</th>
<th>-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

(the indices)
List of ints

Lists can have their elements **set** to a particular value, like with Arrays.

```
  8    9    17    39-100    1092    3    4    -9
```

“set element at index 3 to be 100”
List of ints

You can **get** the value stored at a particular index in a List.

“get the element at index 6”
List of ints

You can **add** an element to the end of a List.

```
| 8 | 9 | 17 | 100 | 1092 | 3 | 4 | -9 | 89 |
```

“add 89 to the end”
List of ints

You can **add** an element at any particular index of a List.

“add 14 at index 2”
List of ints

You can **remove** whatever element lives at a particular index.

```
8  9  14  17  100  1092  3  4  -9  89
```

“remove what’s at index 2”
Defining the List ADT for a List of Strings

```java
public interface List { // List of String class ADT
    // Remove all contents from the list, so it is once again empty
    public void clear();
    // Insert s at the position index in this list.
    // throws IndexOutOfBoundsException - if the index is out of range (index < 0 || index > length)
    public void add(int index, String s);
    // Append s at the end of the list
    public void add(String s);
    // Removes and return the element at the specified position in this list
    // throws IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= length)
    public String remove(int index);
    // Returns the element at the specified position in this list
    // throws IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= length)
    public String get(int index);
    // Returns true if this list contains the specified element. The empty String otherwise
    public boolean contains(String o);
    // Returns the length / number of elements in this list
    public int size();
    // Returns true if this list is empty
    public boolean isEmpty();
}
```
The LinkedList

- A LinkedList is one of many possible ways that we might implement the List ADT
- LinkedLists are made up of Nodes, like we’ve studied
  - The length of a LinkedList is the number of Nodes in the list.
  - The head of a LinkedList is the first Node in the LinkedList.
  - A LinkedList is empty when its head is null.
It’s the same elements before, but now we have an implementation as a LinkedList—a sequence of Nodes pointing each to the next.
List? LinkedList? What’s the difference?

- **List**
  - An interface
  - An abstract data type
  - Specifies what a **List** should be able to do, but not how it works.
  - An ordered collection of elements.

- **LinkedList**
  - An implementation
  - A class
  - One implementation of all the list methods that answers how it works.
  - A chain of Nodes, starting from a head node, that contain each of the elements in a set order.
LinkedList

- Implementation of the List ADT
- Uses Linked Nodes to store data
- Node class is hidden inside the LinkedList class
  - The node class is implemented inside the LinkedList class
  - The Node class is the **inner class** and LinkedList is the **wrapper class**
  - The wrapper class has access to the data fields and methods of the inner class
  - The node class is declared as “private”