In this lab, you will continue with **LinkedBag**.

**LinkedBag**

To begin with, make a **LinkedBag** class with inner class **Node** in the project containing our prior bag work:

```java
public class LinkedBag<T> implements Bag<T> {
    private Node head;

    public class Node {
        public T value;
        public Node next;

        public Node(T v, Node n) {
            value = v;
            next = n;
        }
    }
}
```

See if you can recreate the method **add**, which takes a value and adds it as the first element (the one that **head** refers to). We did this in class Friday and it’s in the online notes, but see if you can write the method yourself before checking it there.

Next write a method **size** that traverses the nodes and counts them. Again, this is something we did in class, but try to write it yourself before checking.

Of course there is more to a list than its size. One method that we haven’t written is **toArray**, which returns an array containing the bag contents. Write this method. First, it should use **size** to allocate an array of the appropriate length. (Look back at **OurBag** for examples of creating an array of type **T[]**.) Then, traverse the nodes using a **while** loop and copy each of the values into an array cell.

We’ve talked about the difficulty of writing test cases for **toArray**; because the bag is not required to store its values in any particular order, the array values could be in any order. One approach is to sort the returned array (i.e. to arrange it into a specific order). Specifically, you can do this with the following lines:

```java
String[] contents = bag.toArray();
Arrays.sort(contents);
```

After these lines, **contents** will contain the values from the bag in **lexicographic order**, which is the same as alphabetic order as long as all the strings in the bag have only lower case letters. Using this trick, design a test case that calls **toArray** and then checks the contents to make sure the bag contains the correct values. (Either check each array cell separately or create an array that has the contents in the correct order and check them with a loop.)

Next write the **remove** method that doesn’t take any arguments. This method can return the value associated with the first node. Again, testing is an issue, but you can at least make sure that once a value is removed (i.e. returned by **remove**), it is no longer in the bag.

If you finish these methods, write/copy the other methods promised by the **Bag** interface and then add some others that seem reasonable. One idea is versions of **add** and **remove** that add or remove the last node rather than the first. These methods would pass the same test cases as the front-oriented ones and wouldn’t be used for **Bag**, but we’ll want these methods for related data structures that we study later in the term.