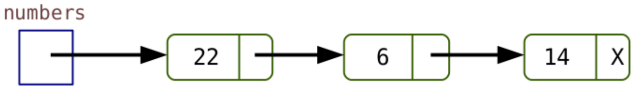
Model 1 Linked Lists

Linked structures "chain" elements using references. Each element of the list is called a ***node***.

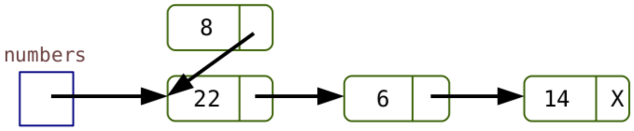
```
public class Node {

    private int value;
    private Node next;
    ...
}
```

```
Node node3 = new Node(14, null);
Node node2 = new Node(6, node3);
Node numbers = new Node(22, node2);
```

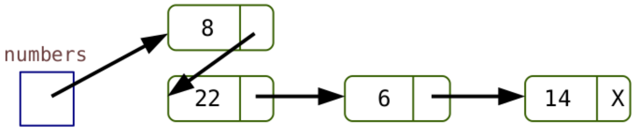


This organization allows fast insertions/deletions near the beginning. For example, to add 8:

```
Node temp = new Node(8, numbers);
```



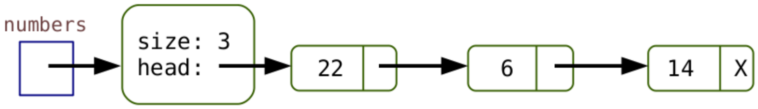
```
numbers = temp;
```



Instead of working with nodes directly, we can design a wrapper class to implement a list:

```
public class MyList {

    private int size;
    private Node head;
    ...
}
```

```
MyList numbers = new MyList();
numbers.addAtStart(14);
numbers.addAtStart(6);
numbers.addAtStart(22);
```



Questions (15 min)

Start time:

1. In `MyList`, how many assignment operations are required to add 14 *at the front* of an empty list? Note that creating a `Node` takes two assignments (one for `value` and one for `next`).

3 operations: 1 to assign the `value`, 1 to assign null to `next`, and 1 to assign the reference of the new Node to the head variable.

2. In `MyList`, how many operations are required to add 22 at the front, after 14 and 6 have been added?

Still just 3, in fact the same as the first insert. Notice that no shifting is required.

3. How many operations are required to add an element *at the end* of `MyList` with 3 elements?

5 operations: 3 to find where the new element goes (by following references to the end of the list), one to create the new node, and one to change the reference of the previous last element.

4. How much memory is needed to store each element in the `LinkedList`? How does that amount compare with using an `ArrayList`?

Linked lists need to store two references per node (one for `value` and one for `next`). In contrast, array lists only need to store the `value` references. At a conceptual level, array lists take up about half as much space as linked lists (not counting empty cells and object overhead).

5. Discuss why `LinkedList` is a poor choice of `List` in the program below.

```java
import java.util.LinkedList;
import java.util.List;

public class LinksAreBad {

    public static void main(String[] args) {
        List<String> list = new LinkedList<>();
        System.out.println("Start");
        addAndGet(list);
        System.out.println("Done!");
    }

    public static void addAndGet(List<String> list) {
        for (int i = 0; i < 1000000; i++) {
            list.add("A");  // add at the end
        }
        for (int i = 0; i < 1000000; i++) {
            list.get(list.size() / 2);  // get the middle
        }
    }
}
```

Each insertion in the middle of the list takes $n/2$ operations (where n is the current size of the list) in order to find the `next` references to assign.

6. If your program requires a `List` collection, how would you decide which implementation to use? (`ArrayList` vs `LinkedList`)

If adding items at the end and accessing random elements, use ArrayList. If inserting items in the middle and only accessing at the two ends, use LinkedList.