18-0: Arrays

- Array Advantages
  - Elements are stored in sequential memory locations
  - Fast lookup of element by index

- Array Disadvantages
  - Difficult to resize the array
  - Adding elements to the middle is expensive
18-1: Linked Lists

- Linked List Advantages
  - Easy to resize a linked list
  - Adding elements to the middle easy

- Linked List Disadvantages
  - Elements are *not* stored in sequential memory locations
  - Finding the $n$th element in the list is slower
public class LinkedListNode {
    int data;
    LinkedListNode next;
}

• This looks very strange – recursive data structure! If a LinkedListNode contains a LinkedListNode, which contains a LinkedListNode, where does it stop?
This looks very strange – recursive data structure! If a LinkedListNode contains a LinkedListNode, which contains a LinkedListNode, where does it stop?

• Remember that class variables are only created when “new” is called – otherwise, you just have a pointer.
public class LinkedListNode
{
    int data;
    LinkedListNode next;
}

What does memory look like when we do this:
LinkedListNode n = new LinkedListNode
What does memory look like when we do this:

```java
LinkedListNode n = new LinkedListNode();
n.data = 3;
n.next = new LinkedListNode();
```
18-6: Linked Lists

![Diagram of linked list structure with objects and next pointers connected in a chain, starting from the head.]
18-7: Linked Lists

- Each element in the list is a *node*
- Each node contains:
  - An Object that represents the data stored in the node
  - A *next* pointer
public class ListNode
{
    public Object data;
    public ListNode next;

    public ListNode(Object d)
    {
        data = d;
        next = null;
    }

    public ListNode(Object d, ListNode n)
    {
        data = d;
        next = n;
    }
}
What would memory look like after the following:

```java
ListNode list = null;
for (int i = 0; i < 5; i++)
{
    list = new ListNode(new Integer(i), list);
}
```
Our linked list class needs to keep track of the *head* of the list

All other elements are reachable from the head
  - Follow the next pointers
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    // Methods to manipulate the list
}
Linked List Methods

- **insert(Object o)** Insert an element at the front of the list
- **get(int index)** Get an element at a specified index
- **last()** Get the last element in the list
public class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public void insert(Object o) {
    }
}
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public void insert(Object o)
    {
        ListNode newElem = new ListNode(o);
        newElem.next = head;
        head = newElem;
    }
}
What if we changed the order a little bit?

```java
class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public void insert(Object o) {
        ListNode newElem = new ListNode(o);
        head = newElem;
        newElem.next = head;
    }
}
```
What if we changed the order a little bit?

```java
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public void insert(Object o)
    {
        ListNode newElem = new ListNode(o); // DOES NOT WORK!!
        head = newElem;
        newElem.next = head;
    }
}
```
We can use constructors to make our life easier ...

```java
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public void insert(Object o)
    {
        head = new ListNode(o, head);
    }
}
```
Object get(int index)

public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public Object get(int index) { ... }
}
18-19: Linked Lists

- **Object get(int index)**

```java
public class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public Object get(int index) {
        ListNode tmp = head;
        for (int i = 0; i < index; i++)
            tmp = tmp.next;
        return tmp.data;
    }
}
```
Object `get(int index)`

```java
public class LinkedList {
  private ListNode head;
  public LinkedList() {
    head = null;
  }
  public Object get(int index) {
    ListNode tmp = head;
    for (int i = 0; i < index; i++) {
      if (tmp == null) // Added some error checking ...
        return null;
      tmp = tmp.next;
    }
    return tmp.data;
  }
}
```
• Object get(int index)
  • Can we do this recursively?

```java
public class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public Object get(int index, ListNode list) {
    }

    public Object get(int index) {
        return get(index, head);
    }
}
```
public class LinkedList
{
    private ListNode head;

    public LinkedList()
    {
        head = null;
    }

    public Object get(int index, ListNode list)
    {
        if (index == 0)
        {
            return list.data;
        }
        return get(index - 1, list.next);
    }

    public Object get(int index)
    {
        return get(index, head);
    }
}
Object last()  
Return the last element of the list (leaving list unchanged)
public class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public Object last() {
        ListNode tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        return tmp.data;
    }
}

18-24: Linked Lists
Download LinkedListNode, LinkedList, LinkedListDriver from website

Implement the methods (ordered from easiest to hardest)

- deleteFirst
- insertAtEnd
- deleteLast