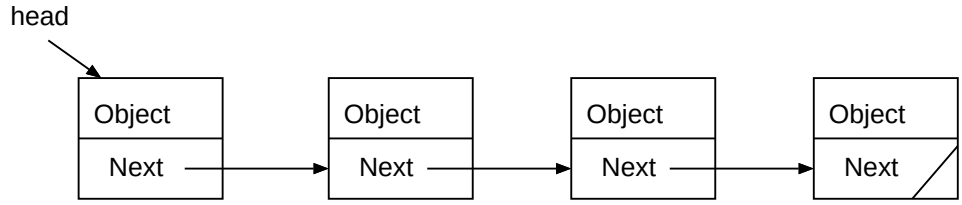
18-0: **Linked Lists**

- Linked List node
  - Data
  - Pointer to the next element in the linked list
- Keep a pointer to the first element of the list

18-1: **Linked List Node**

```
public class LinkedListNode
{
   int data;
   LinkedListNode next;
}

LinkedListNode head;
```

18-2: **Linked Lists**



18-3: **Create list 1-10**

```
public class LinkedListNode
{
   int data;
   LinkedListNode next;
}
```

- Create a linked list that contains the elements 1-10
- (example on board)

18-4: **Create list 1-10**

```
LinkedListNode list = null;
for (int i = 10; i <= 1; i--)
{
    LinkedListNode tmp = new LinkedListNode();
    tmp.next = list;
    tmp.data = i;
    list = tmp;
}
```

18-5: **Linked Lists**

```
LinkedListNode list = null;
// ... code to add elements to the linked list
```

- Code to remove the 2nd element from the linked list:

18-6: **Linked Lists**

```
LinkedListNode list = null;
// ... code to add elements to the linked list
```

- Code to remove the 2nd element from the linked list:

```
list.next = list.next.next;
```

18-7: **Linked List Class**

```
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;
   }
}
```

18-8: **Linked Lists**

- `Object get(int index)`

```
public class LinkedList
{
   private ListNode head;

   public LinkedList()
   {
      head = null;
   }

   public Object get(int index)
   {
     // fill me in!
   }
}
```

18-9: **Linked Lists**

- `Object get(int index)`

```
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;
   }
}
```

18-10: **Practical Experience Review**

- deleteFirst
- insertAtEnd
- deleteLast

18-11: **Practical Experience Review**

```
public class LinkedList
{
   private ListNode head;

   public LinkedList()
   {
      head = null;
   }
```

```
   public Object deleteFirst()
   {
      if (head == null)
      {
         throw new NoSuchElementException();
      }
      Object returnVal = tmp.data;
      head = head.next;
      return returnVal;
   }
}
```

18-12: **Practical Experience Review**

```
   public void insertAtEnd(Object elem)
   {
      if (head == null)
      {
         head = new LinkedListNode(elem, null);
      }
      else
      {
           ListNode tmp = head;
           while (tmp.next != null)
           {
              tmp = tmp.next;
           }
           tmp.next = new LinkedListNode(elem, null);
      }
   }
```

18-13: **Practical Experience Review**

```
   public Object deleteLast()
   {
      if (head == null)
      {
         throw new NoSuchElementException();
      }
      else if (head.next == null)
      {
          Object returnVal = tmp.data;
          head = null;
          return returnVal;
      } else {
         LinkedListNode tmp = head;
         while (tmp.next.next != null)
         {
            tmp = tmp.next;
         }
          Object returnVal = tmp.next.data;
          tmp.next = null;
          return returnVal;
      }
   }
```

18-14: **Linked Lists**

- Linked List Methods
  - `void set(int index, Object o)` Set the ith element in the list to be o

18-15: **Linked Lists**

```
public class LinkedList
{
   private ListNode head;

   public LinkedList()
   {
      head = null;
   }

   // NOTE:  No error checking done here!
   void set(int index, Object o)
   {
      ListNode tmp = head;
      for (int i = 0; i < index; i++)
      {
         tmp = tmp.next;
      }
      tmp.daa = o;
   }
}
```

18-16: **Linked Lists**

- Finding an element in a list
- `boolean find(ListNode list, Object o)`

18-17: **Linked Lists**

```
public boolean find(ListNode list, Object o)
{
   while (list != null)
   {
      if (list.data.equals(o))
      {
         return true;
      }
      list = list.next;
   }
   return false;
}
```

- Recursive?

18-18: **Linked Lists**

```
public boolean find(ListNode list, Object o)
{
   if (list == null)
      return false;
   if (list.data.equals(o))
      return true;
   return find(list.next, o);
}
```

18-19: **Linked Lists**

```
public class LinkedList
{
   private ListNode head;

   public LinkedList()
   {
      head = null;
   }

   boolean find(Object o) { ... }
}
```

18-20: **Linked Lists**

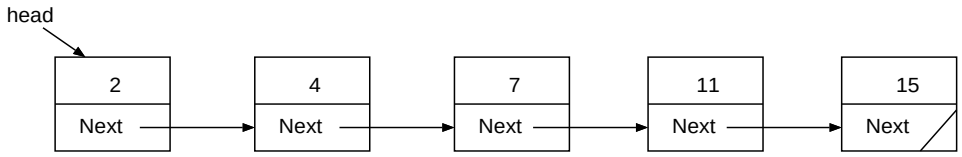
```
public class LinkedList
{
   private ListNode head;

   public LinkedList()
   {
      head = null;
   }

   boolean find(Object o)
   {
      ListNode tmp = head;
      while (tmp != null)
      {
         if (tmp.data.equals(o))
            return true;
         tmp = tmp.next;
      }
      return false;
   }
}
```

18-21: **Linked Lists**

- Linked List Methods
  - `remove(Object o)` Remove the first occurrence of an object from a list
    - If we advance a pointer until we find the element to remove, we've gone too far
    - Still need a special case for deleting the first element in the list

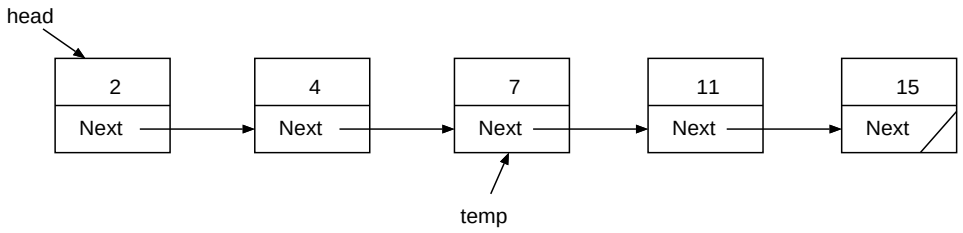
18-22: **Linked Lists**

Deleting the 7 from the list



18-23: **Linked Lists**

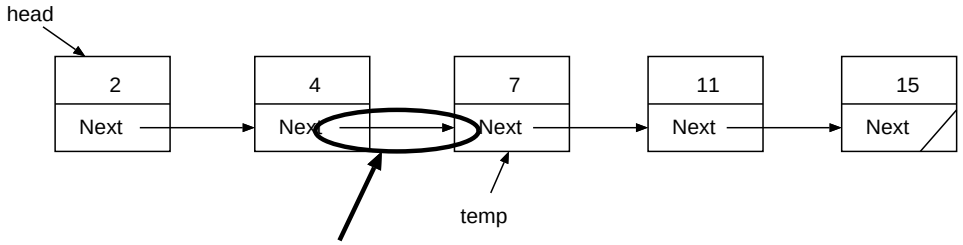
Deleting the 7 from the list



Advace a temp pointer until it points to the element to be deleted

18-24: **Linked Lists**

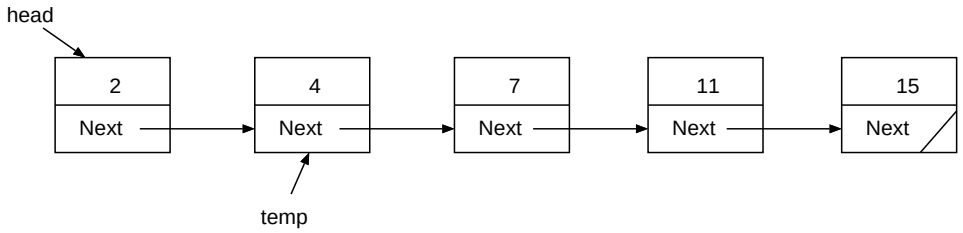
Deleting the 7 from the list



Need to change this pointer to remove the 7,
no way to get to it from temp

18-25: **Linked Lists**

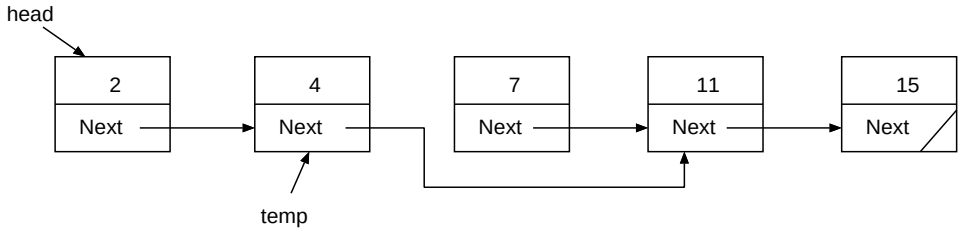
Deleting the 7 from the list



If we stop one step before we get to the 7,
we can remove the from the list

18-26: **Linked Lists**

Deleting the 7 from the list



If we stop one step before we get to the 7,
we can remove the from the list

18-27: **Practical Experience**

- Dowmload LinkedList, ListNode, and LinkedListDriver classes from website
- Fill in missing classes removeAt, remove, removeAll