Linked Lists

The contiguous nature of the memory allocated for arrays means that resizing and shifting is required.

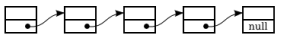
An alternative is to allocate memory one *node* at a time. Each node contains a reference to the next node.

The statement A = new int[5]; creates an array that holds five elements of type int. A is a name for the whole array.

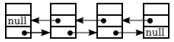
A:

A.length:	(5)
A[0]:	0
A[1]:	0
A[2]:	0
A[3]:	0
A[4]:	0

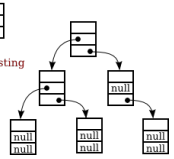
The array contains five elements, which are referred to as A[0], A[1], A[2], A[3], A[4]. Each element is a variable of type int. The array also contains A.length, whose value cannot be changed.



When an object contains a reference to an object of the same type, then several objects can be linked together into a list. Each object refers to the next object.



Things get even more interesting when an object contains two references to objects of the same type. In that case, more complicated data structures can be constructed.



ListNode

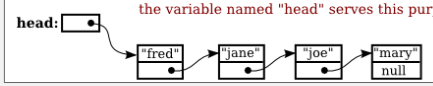
True or false: the `Node` class is part of the standard Java library, like `String` or `Scanner`.

	Answer	Respondents	Percentage
✗	True	2	15%
✓	False	11	85%

What will be printed when the following code is executed using the linked list shown?

```
for ( Node runner = head ; runner.next != null ; runner = runner.next ) {
   System.out.println(runner.item);
}
```

For a list to be useful, there must be a variable that points to the first node in the list. Here, the variable named "head" serves this purpose.




```
class Node {
   String item;
   Node next;
}
```

	Answer	Respondents	Percentage
✗	fred	2	15%
✗	jane	0	0%
✓	joe	3	23%
✗	mary	1	8%
✗	null	1	8%
✗	none of the above	6	46%

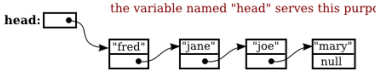
intent of question was what will be the *last* thing printed

right answer given question as written

What will be printed when the following code is executed using the linked list shown?

```
int count = 0;
for ( Node runner = head ; runner != null ; runner = runner.next, count++ ) {}
System.out.println(count);
```

For a list to be useful, there must be a variable that points to the first node in the list. Here, the variable named "head" serves this purpose.




```
class Node {
   String item;
   Node next;
}
```

	Answer	Respondents	Percentage
✗	0	0	0%
✗	1	0	0%
✗	2	0	0%
✗	3	3	23%
✓	4	8	62%
✗	5	0	0%
✗	something else	0	0%
✗	nothing - this isn't legal code	2	15%

Questions

- `printReversed` and `addItemsInList`…?

```
/**
 *  Compute the sum of all the integers in a linked list of integers.
 *  @param head a pointer to the first node in the linked list
 */
public static int addItemsInList( IntNode head ) {
   if ( head == null ) {
         // Base case:  The list is empty,  so the sum is zero.
      return 0;
   }
   else {
         // Recursive case:  The list is non-empty.  Find the sum of
         // the tail list, and add that to the item in the head node.
         // (Note that this case could be written simply as
         //     return head.item + addItemsInList( head.next );)
      int listsum = head.item;
      int tailsum = addItemsInList( head.next );
      listsum = listsum + tailsum;
      return listsum;
   }
}
```

these use a technique called *recursion* which we will see later

```
public static void printReversed( Node head ) {
   if ( head == null ) {
         // Base case:  The list is empty, and there is nothing to print.
      return;
   }
   else {
         // Recursive case:  The list is non-empty.
      printReversed( head.next );  // Print strings from tail, in reverse order.
      System.out.println( head.item );  // Then print string from head node.
   }
}
```


`ListNode`

- most programming languages provide direct support for arrays, but not for linked lists (or other linked structures)
- many variations of the idea

→ write your own class representing a list node

```
public class ListNode {
  private int elt_;   // element stored in the node
  private ListNode next_;   // next node

  public ListNode ( int elt ) {
    elt_ = elt;
    next_ = null;
  }
  public ListNode ( int elt, ListNode next ) {
    elt_ = elt;
    next_ = next;
  }
  public int getElt () { return elt_; }
  public ListNode getNext () { return next_; }
  public void setNext ( ListNode next ) {
    next_ = next;
  }
}
```


Working With Linked Lists

Strategy –

- draw before and after pictures for a typical example
  - avoid common special cases like empty or 1-node list, or working with first or last node
- identify what is different between the two pictures
  - new node objects?
  - different next values in existing node objects? (e.g. changed links)
- get fingers pointing to the nodes of interest
  - new variables, not new objects
- do the steps to transform the before picture into the after picture
  - create new nodes
  - use list node operations (`setNext`) to change links in existing nodes

Working With Linked Lists

Strategy continued –

- repeat for (legal) special cases
  - identify where the code fails to handle them (if it fails)
  - include an `if` before that point to catch and divert that case to code that handles it