Announcements

ICLICKER QUESTION

What is the last node in the following linked list?

```java
node2.setNext(node1);
node4.setNext(node2);
node3.setNext(node4);
node1.setNext(node3);
```

A. 1
B. 2
C. 3
D. 4
E. none
Structuring Music

Linked List Traversal

```java
public Phrase getPhrase()
{
    NoteNode current = this;
    Phrase phr = new Phrase();
    while (current != null)
    {
        phr.addNote(current.note);
        current = current.next;
    }
    return phr;
}
```

Version 2: Note Linked List

NoteNode current = this;

while (current != null)
{
    current = current.next;
}
Traversing arrays vs. lists

```
// Traverse the Linked List
public Phrase getPhrase()
{
    NoteNode current = this;
    while(current != null)
    // true
        current = current.next;
    // false
    return phr;
}

// Traverse the Array
public Phrase getPhrase()
{
    Phrase phr = new Phrase();
    for(int i = 0; i < notes.size(); i++)
    phr.addNote(notes.get(i));
    return phr;
}
```

Inner Classes

- So far we have been defining methods within the node class.
- We can create a class to manage the entire list.
  - `NoteLinkedList`
- Will contain a reference to the **head** of the list.
- Where should the node class be defined?
  - Only used in `NoteLinkedList`
  - Can be defined locally.
    - Inner Class
public class NoteLinkedList
{
    private static class Node
    {
        //---// Node Fields //---//
        private Note note;
        private Node next;
    }
    //---//---// Fields //---//---//
    private Node head;
    //---//---// Constructors //---//---//
    public NodeLinkedList()
    {
        this.head = null;
    }
    //---//---// Methods //---//---//
    private Node getLast()
    {
        Node cur = head;
        while(cur != null && cur.next != null)
            cur = cur.next;
        return cur;
    }
    public void add(Note note)
    {
        if(head == null)
            head = new Node(note);
        else
            getLast().next = new Node(note);
    }
}
Inserting into lists

```java
n3.next = n2;
n1.next = n3;
```

Insert After

```java
public void insertAfter(Node toInsert) {
    toInsert.next = this.next;
    this.next = toInsert;
}
```

Inserting into arrays

- Be sure there is room (or you need to create a larger array)
- Move everything from position to end up by one spot
- Assign to current position.

Why use linked lists versus arrays?

- Just two reasons now, more later:
  1. Can grow to any size (well, as long as memory permits)
     - Just create a new element and poke it into the list.
  2. MUCH easier to insert!
     - Look at how easily we put part3 between part1 and part2.
More on Arrays vs. Lists

- **Arrays**
  - Easier to traverse
  - Very fast to access a specific \( n^{th} \) element
  - But really a pain to insert and delete.
    - Hard to write the code
    - Can take a long time if it’s a big array

- **Lists**
  - More complex to traverse
  - Slower to access a specific element
  - Very easy to insert (and later we’ll see, delete)
    - Simple code
    - Fast

---

**ICLICKER QUESTION**

Which operation is faster on an Array than on a Linked List?

A. Get (an element)
B. Insert
C. Add (to the end)
D. Remove
E. None