Announcements

STRUCTURING MUSIC

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

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

// Traverse the Linked List
public Phrase getPhrase() {
    NoteNode current = this;
    Phrase phr = new Phrase();
    while (current != null) {
        phr.addNote(current.note);
        current = current.next;
    }
    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;
}

Version 3: Note Linked List

public class NoteLinkedList {
    private static class Node {
        //---// Node Fields //---//
        private Note note;
        private Node next;
    }
}
//----// Node Constructors //----//
public Node(Note note)
{
  this.note = note;
}

public Node(Note note, Node next)
{
  this.note = note;
  this.next = next;
}
} //----// end class Node //----//

//----//------// Fields //----//------//
private Node head;

//----//------// Constructors //----//------//
public NoteLinkedList()
{
  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);
}
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?

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
Copy

- Two types of copy for objects:
  - Shallow Copy
    - Copy only the reference to the object
    - Two references to the same object
  - Deep Copy
    - Copy the actual data

Deep Copy

```
private static class Node {
    //...

    public Node copy() {
        return new Node(note.copy(), next);
    }
}
```

Repeat

```
public void repeat() {
    Node head2 = head.copy();
    Node cur = head;
    Node cur2 = head2;
    while (cur.next != null) {
        cur2.next = cur.next.copy();
        cur = cur.next;
        cur2 = cur2.next;
    }
    cur.next = head2;
}
```