11-0: Arrays

Previously we talked about using arrays to store sequences of data.

Advantages:
- Everything is stored in sequential memory locations.
- Fast lookup of elements.

Disadvantages:
- Hard to resize
- Removing or adding an element in the middle is costly.

11-1: Linked Lists

Linked lists have the opposite advantages and disadvantages

Advantages:
- Easy to insert and remove
- Easy to resize

Disadvantages:
- Elements are not stored sequentially
- Finding the nth element is slower.

11-2: Linked Lists

The general idea:
- Each element of the list will “know” who the next element is.
- Let’s try this as a class.

11-3: List elements

So how do we do this in Java?

Our Element class needs to have two components:
- The data that we want to store in the list.
- A pointer to the next element in the list.

```java
public class ListItem {
    public Object data;
    public ListItem next;
    public ListItem(Object d) {
        data = d;
        next = null;
    }
}
```

11-4: List elements
11-5: Arranging ListItems

- ListItems hook together like a chain.
- All we need to do is keep track of the beginning of the chain.
- No need to allocate everything ahead of time.

11-6: The LinkedList class

- The LinkedList will be responsible for hanging onto the 'head' of the list and providing methods for working with the list.
  - insert()
  - insertAt(index)
  - get(index)
  - remove(index)
  - find(object)

11-7: The LinkedList class

```java
public class LinkedList {
    public ListItem head = null;
    public void insert(Object o) { ... }
    public void insertAt(Object o, int index) { ... }
    public Object get(int index) { ... }
    public Object remove(int index) { ... }
    public int find(Object o) { ... }
}
```

11-8: Adding

- So how do we add something to the front of a linked list?
  - Have the new thing point to the currently-first ListItem
  - Point 'head' to our new thing.

```java
public void insert(Object o) {
    ListItem l = new ListItem(o);
    l.next = head;
    head = l;
}
```

11-9: Adding

- What about adding something into the middle of a list?
- If we want an item to go between current elements 5 and 6, then we need our new item to point to 6, and 5 to point to the new element.
- How do we code this?

11-10: Adding

```java
public void insertAt(Object o, int index) {
    // first find the place to insert it.
    ListItem pointer = head;
    ListItem l = new ListItem(o);
    for (int i = 0; i < index - 1; i++) {
        pointer = pointer.next;
    }
    l.next = pointer.next;
    pointer.next = l;
}
11-11: Adding

- Are there any special cases we need to worry about?

11-12: Adding

- Are there any special cases we need to worry about?
  - What if the list is empty?
  - What if it only has one element?
  - What if we're adding at the end?

11-13: Exercise

- Code the ListItem and LinkedList classes as described above.
- Write a main method that creates a list and prompts the user for strings, which are always added to the front of the list.
- Add a method to the LinkedList class called toString().
- This should walk the list and print out each of the strings in order.