CONTAINERS AND VARIABLES

- in OOP, objects cannot float around without some form of attachment or containment
- either ‘attach’ an object to a variable
  ```java
  Obj o = new Obj();  o = Obj()
  ```
- and pass it around by reference
  ```java
  Obj o2, o3;  o2 = o3 = None
  o2 = o;  o2 = o
  o3 = o;  o3 = o
  ```
- or … ‘put’ the object in some class that can hold more than one instance
  ```java
  Obj o = new Obj();
  Container c = new Container();
  c.add(o);
  Obj o2 = c.getNext();
  ```
CONTAINERS FOR HOLDING OBJECTS

- what sort of object is:
  - a bag of rocks?
  - a horde of monsters?
  - the stash of weapons a player is carrying?
  - the suite of rooms in a game?

- does a bag of rocks extend Rock?
- does a horde of monsters extend Monster?
- does a stash of weapons extend Weapon?
- does a suite of rooms extend Room?

- think of bags, hordes, stashes, and suites as objects in their own right.

- collection, or container, objects
- what operations would you perform on such an object?
- what would be its responsibilities?
A VARIETY OF CONTAINERS

- ordered (sequenced) versus unordered
  - linear ordering (queue, stack, list) [physical examples?]
  - branching (tree, graph) [physical examples?]
  - fixed vs. dynamic (sorted, rebalanced, evolving topology) [physical examples?]
- homogeneous (all contents of same type) vs. heterogeneous [physical examples?]
  - e.g., arrays
  - arbitrary type (junk drawer)
- what common operations (methods) on containers?
  - initializing
  - adding/removing elements
  - enumerating each an element for ‘scrutiny’
  - count of contents
  - accepting visitors?
- why put objects in containers?
  - to organize by type
  - to organize by use to be made of them
  - to implement notion of ownership
  - to filter, catalog, sort, arrange, ...
ITERATION OVER (UNORDERED) SETS

- a useful class of container would be one that holds elements in no particular order, and which allows you access to them one-at-a-time, without removing them.
- internally, the items are inevitably in some order.

- simulating randomness through the enumeration process
  - each element accessed once and only once
  - different order of enumeration each time

- how to interleave add/remove/enumeration of elements?
PYTHON’S LIST CLASS

- lists in Python
  ```python
define symbols
listA = []
listB = listA
listC = listB[2:4]
for item in listC:
  print item
```  
- lists implement the standard sequence interface
  ```python
define symbols
len(listA) # length
item = listA[0] # selected item at given index
sequence = listA[0 : 3] # select sublist in range
```  
- is a list a container class?
  - it contains things, right?
  - what does it take to be a ‘container’ ???
    assisting users in common activities (adding, removing, enumerating)
## Method Summary

<table>
<thead>
<tr>
<th>boolean</th>
<th>add(Object o)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ensures that this collection contains the specified element (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>addAll(Collection c)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adds all of the elements in the specified collection to this collection (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>clear()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Removes all of the elements from this collection (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>contains(Object o)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns true if this collection contains the specified element.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>containsAll(Collection c)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns true if this collection contains all of the elements in the specified collection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>equals(Object o)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compares the specified object with this collection for equality.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>hashCode()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the hash code value for this collection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>isEmpty()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns true if this collection contains no elements.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iterator</th>
<th>iterator()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns an iterator over the elements in this collection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>remove(Object o)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Removes a single instance of the specified element from this collection, if it is present (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>removeAll(Collection c)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Removes all this collection's elements that are also contained in the specified collection (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean</th>
<th>retainAll(Collection c)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retains only the elements in this collection that are contained in the specified collection (optional operation).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>size()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the number of elements in this collection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object[]</th>
<th>toArray()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns an array containing all of the elements in this collection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object[]</th>
<th>toArray(Object[] a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.</td>
</tr>
</tbody>
</table>
### Constructor Summary

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected</td>
<td><code>AbstractCollection()</code></td>
</tr>
<tr>
<td></td>
<td>Sole constructor.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td><code>add(Object o)</code></td>
<td>Ensures that this collection contains the specified element (optional operation).</td>
</tr>
<tr>
<td>boolean</td>
<td><code>addAll(Collection c)</code></td>
<td>Adds all of the elements in the specified collection to this collection (optional operation).</td>
</tr>
<tr>
<td>void</td>
<td><code>clear()</code></td>
<td>Removes all of the elements from this collection (optional operation).</td>
</tr>
<tr>
<td>boolean</td>
<td><code>contains(Object o)</code></td>
<td>Returns true if this collection contains the specified element.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>containsAll(Collection c)</code></td>
<td>Returns true if this collection contains all of the elements in the specified collection.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>isEmpty()</code></td>
<td>Returns true if this collection contains no elements.</td>
</tr>
<tr>
<td>abstract</td>
<td><code>Iterator iterator()</code></td>
<td>Returns an iterator over the elements contained in this collection.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>remove(Object o)</code></td>
<td>Removes a single instance of the specified element from this collection, if it is present (optional operation).</td>
</tr>
<tr>
<td>boolean</td>
<td><code>removeAll(Collection c)</code></td>
<td>Removes from this collection all of its elements that are contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td>boolean</td>
<td><code>retainAll(Collection c)</code></td>
<td>Retains only the elements in this collection that are contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td>int</td>
<td><code>size()</code></td>
<td>Returns the number of elements in this collection.</td>
</tr>
<tr>
<td>Object[]</td>
<td><code>toArray()</code></td>
<td>Returns an array containing all of the elements in this collection.</td>
</tr>
<tr>
<td>Object[]</td>
<td><code>toArray(Object[] a)</code></td>
<td>Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.</td>
</tr>
<tr>
<td>String</td>
<td><code>toString()</code></td>
<td>Returns a string representation of this collection.</td>
</tr>
</tbody>
</table>

**Methods inherited from class java.lang.Object**
- `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`  

**Methods inherited from interface java.util.Collection**
- `equals`, `hashCode`
## Constructor Summary

<table>
<thead>
<tr>
<th>protected</th>
<th>AbstractCollection()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sole constructor.</td>
<td></td>
</tr>
</tbody>
</table>

## Method Summary

### AbstractCollection

#### protected

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(Object o)</td>
<td>Ensures that this collection contains the specified element (optional operation).</td>
</tr>
<tr>
<td>addAll(Collection c)</td>
<td>Adds all of the elements in the specified collection to this collection (optional operation).</td>
</tr>
<tr>
<td>clear()</td>
<td>Removes all of the elements from this collection (optional operation).</td>
</tr>
<tr>
<td>contains(Object o)</td>
<td>Returns true if this collection contains the specified element.</td>
</tr>
<tr>
<td>containsAll(Collection c)</td>
<td>Returns true if this collection contains all of the elements in the specified collection.</td>
</tr>
<tr>
<td>equals(Object o)</td>
<td>Compares the specified object with this collection for equality.</td>
</tr>
<tr>
<td>hashCode()</td>
<td>Returns the hash code value for this collection.</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>Returns true if this collection contains no elements.</td>
</tr>
<tr>
<td>iterator()</td>
<td>Returns an Iterator over the elements in this collection.</td>
</tr>
<tr>
<td>remove(Object o)</td>
<td>Removes a single instance of the specified element from this collection, if it is present (optional operation).</td>
</tr>
<tr>
<td>removeAll(Collection c)</td>
<td>Removes all of this collection's elements that are also contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td>retainAll(Collection c)</td>
<td>Retains only the elements in this collection that are contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td>size()</td>
<td>Returns the number of elements in this collection.</td>
</tr>
<tr>
<td>toObject()</td>
<td>Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of Object[].</td>
</tr>
<tr>
<td>toArray(Object[] a)</td>
<td>Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of Object[].</td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object:

copy(), equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.util.Collection:

equals, hashCode
**CLASS** AbstractList **EXTENDS** AbstractCollection

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
</table>
| **void** add(int index, Object element)  
Inserts the specified element at the specified position in this list (optional operation). |
| **boolean** add(Object o)  
Appends the specified element to the end of this List (optional operation). |
| **boolean** addAll(int index, Collection c)  
Inserts all of the elements in the specified collection into this list at the specified position (optional operation). |
| **void** clear()  
Removes all of the elements from this collection (optional operation). |
| **boolean** equals(Object o)  
Compares the specified object with this list for equality. |
| **abstract Object** get(int index)  
Returns the element at the specified position in this list. |
| **int** hashCode()  
Returns the hash code value for this list. |
| **int** indexOf(Object o)  
Returns the index in this list of the first occurrence of the specified element, or -1 if the list does not contain this element. |
| **Iterator** iterator()  
Returns an iterator over the elements in this list in proper sequence. |
| **int** lastIndexOf(Object o)  
Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| **ListIterator** listIterator()  
Returns an iterator of the elements in this list (in proper sequence). |
| **ListIterator** listIterator(int index)  
Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in the list. |
| **Object** remove(int index)  
Removes the element at the specified position in this list (optional operation). |
| **protected void** removeRange(int fromIndex, int toIndex)  
Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| **Object** set(int index, Object element)  
Replaces the element at the specified position in this list with the specified element (optional operation). |
| **List** subList(int fromIndex, int toIndex)  
Returns a view of the portion of this list between fromIndex, inclusive, and toIndex, exclusive. |

**Methods inherited from class java.util.AbstractCollection**

addAll, contains, containsAll, isEmpty, remove, removeAll, retainAll, size, toArray, toArray, toString
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(int index, Object element)</code></td>
<td>Inserts the specified element at the specified position in this list (optional operation).</td>
</tr>
<tr>
<td><code>add(Object o)</code></td>
<td>Appends the specified element to the end of this list (optional operation).</td>
</tr>
<tr>
<td><code>addAll(Collection c)</code></td>
<td>Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).</td>
</tr>
<tr>
<td><code>addAll(int index, Collection c)</code></td>
<td>Inserts all of the elements in the specified collection into this list at the specified position (optional operation).</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>Removes all of the elements from this list (optional operation).</td>
</tr>
<tr>
<td><code>contains(Object o)</code></td>
<td>Returns true if this list contains the specified element.</td>
</tr>
<tr>
<td><code>containsAll(Collection c)</code></td>
<td>Returns true if this list contains all of the elements of the specified collection.</td>
</tr>
<tr>
<td><code>equals(Object o)</code></td>
<td>Compares the specified object with this list for equality.</td>
</tr>
<tr>
<td><code>get(int index)</code></td>
<td>Returns the element at the specified position in this list.</td>
</tr>
<tr>
<td><code>hashCode()</code></td>
<td>Returns the hash code value for this list.</td>
</tr>
<tr>
<td><code>indexOf(Object o)</code></td>
<td>Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.</td>
</tr>
<tr>
<td><code>isEmpty()</code></td>
<td>Returns true if this list contains no elements.</td>
</tr>
<tr>
<td><code>iterator()</code></td>
<td>Returns an iterator over the elements in this list in proper sequence.</td>
</tr>
<tr>
<td><code>lastIndexOf(Object o)</code></td>
<td>Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.</td>
</tr>
<tr>
<td><code>listIterator()</code></td>
<td>Returns a list iterator of the elements in this list (in proper sequence).</td>
</tr>
<tr>
<td><code>listIterator(int index)</code></td>
<td>Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.</td>
</tr>
<tr>
<td><code>remove(int index)</code></td>
<td>Removes the element at the specified position in this list (optional operation).</td>
</tr>
<tr>
<td><code>remove(Object o)</code></td>
<td>Removes the first occurrence in this list of the specified element (optional operation).</td>
</tr>
<tr>
<td><code>removeAll(Collection c)</code></td>
<td>Removes from this list all the elements that are contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td><code>retainAll(Collection c)</code></td>
<td>Retains only the elements in this list that are contained in the specified collection (optional operation).</td>
</tr>
<tr>
<td><code>set(int index, Object element)</code></td>
<td>Replaces the element at the specified position in this list with the specified element (optional operation).</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>Returns the number of elements in this list.</td>
</tr>
<tr>
<td><code>subList(int fromIndex, int toIndex)</code></td>
<td>Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>Returns an array containing all of the elements in this list in proper sequence.</td>
</tr>
<tr>
<td><code>toArray(Object[] a)</code></td>
<td>Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.</td>
</tr>
</tbody>
</table>
AND INTRODUCING THE List INTERFACE

- 24 methods
- that’s a lot!

- add(Object o)
- clear()
- contains(Object o)
- get(int index)
- indexOf(Object o)
- isEmpty()
- iterator()
- ListIterator(int index)
- remove(int index)
- remove(Object o)
- set(int index, Object o)
- size()

- add(int index, Object o)
- addAll(Collection c)
- addAll(int index, Collection c)
- containsAll(Collection c)
- equals(Object o)
- hashCode()
- lastIndexOf(Object o)
- removeAll(Collection c)
- retainAll(Collection c)
- subList(int fromIndex, int toIndex)
- toArray()
- toArray(Object[] a)
### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayList()</code></td>
<td>Constructs an empty list with an initial capacity of ten.</td>
</tr>
<tr>
<td><code>ArrayList(Collection c)</code></td>
<td>Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.</td>
</tr>
<tr>
<td><code>ArrayList(int initialCapacity)</code></td>
<td>Constructs an empty list with the specified initial capacity.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(int index, Object element)</code></td>
<td>Inserts the specified element at the specified position in this list.</td>
</tr>
<tr>
<td><code>add(Object o)</code></td>
<td>Appends the specified element to the end of this list.</td>
</tr>
<tr>
<td><code>addAll(Collection c)</code></td>
<td>Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's iterator.</td>
</tr>
<tr>
<td><code>addAll(int index, Collection c)</code></td>
<td>Inserts all of the elements in the specified Collection into this list, starting at the specified position.</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>Removes all of the elements from this list.</td>
</tr>
<tr>
<td><code>clone()</code></td>
<td>Returns a shallow copy of this <code>ArrayList</code> instance.</td>
</tr>
<tr>
<td><code>contains(Object elem)</code></td>
<td>Returns true if this <code>ArrayList</code> contains the specified element.</td>
</tr>
<tr>
<td><code>ensureCapacity(int minCapacity)</code></td>
<td>Increases the capacity of this <code>ArrayList</code> instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.</td>
</tr>
<tr>
<td><code>get(int index)</code></td>
<td>Returns the element at the specified position in this list.</td>
</tr>
<tr>
<td><code>indexOf(Object elem)</code></td>
<td>Searches for the first occurrence of the given argument, testing for equality using the equals method.</td>
</tr>
<tr>
<td><code>isEmpty()</code></td>
<td>Tests if this list has no elements.</td>
</tr>
<tr>
<td><code>lastIndexOf(Object elem)</code></td>
<td>Returns the index of the last occurrence of the specified object in this list.</td>
</tr>
<tr>
<td><code>remove(int index)</code></td>
<td>Removes the element at the specified position in this list.</td>
</tr>
<tr>
<td><code>removeRange(int fromIndex, int toIndex)</code></td>
<td>Removes from this list all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.</td>
</tr>
<tr>
<td><code>set(int index, Object element)</code></td>
<td>Replaces the element at the specified position in this list with the specified element.</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>Returns the number of elements in this list.</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>Returns an array containing all of the elements in this list in the correct order.</td>
</tr>
<tr>
<td><code>toArray(Object[] a)</code></td>
<td>Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.</td>
</tr>
<tr>
<td><code>trimToSize()</code></td>
<td>Trims the capacity of this <code>ArrayList</code> instance to be the list's current size.</td>
</tr>
</tbody>
</table>
ArrayList EXTENDS AbstractList
IMPLEMENTS Collection, List

- `add(int index, Object o)`
- `add(Object o)`
- `clear()`
- `contains(Object o)`
- `get(int index)`
- `indexOf(Object o)`
- `isEmpty()`
- `remove(int index)`
- `set(int index, Object o)`
- `size()`

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ArrayList()</strong></td>
</tr>
<tr>
<td>Constructs an empty list with an initial capacity of ten.</td>
</tr>
<tr>
<td><strong>ArrayList(Collection c)</strong></td>
</tr>
<tr>
<td>Constructs a list containing the elements of the specified collection, in</td>
</tr>
<tr>
<td>the order they are returned by the collection's iterator.</td>
</tr>
<tr>
<td><strong>ArrayList(int initialCapacity)</strong></td>
</tr>
<tr>
<td>Constructs an empty list with the specified initial capacity.</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add(int index, Object element)</strong></td>
</tr>
<tr>
<td>Inserts the specified element at the specified position in this list.</td>
</tr>
<tr>
<td><strong>add(Object element)</strong></td>
</tr>
<tr>
<td>Appends the specified element to the end of this list.</td>
</tr>
<tr>
<td><strong>addAll(Collection c)</strong></td>
</tr>
<tr>
<td>Appends all of the elements in the specified Collection to the end of</td>
</tr>
<tr>
<td>this list, in the order that they are returned by the specified</td>
</tr>
<tr>
<td>Collection's iterator.</td>
</tr>
<tr>
<td><strong>addAll(int index, Collection c)</strong></td>
</tr>
<tr>
<td>Inserts all of the elements in the specified Collection into this list,</td>
</tr>
<tr>
<td>starting at the specified position.</td>
</tr>
<tr>
<td><strong>clear()</strong></td>
</tr>
<tr>
<td>Removes all of the elements from this list.</td>
</tr>
<tr>
<td><strong>clone()</strong></td>
</tr>
<tr>
<td>Returns a shallow copy of this ArrayList instance.</td>
</tr>
<tr>
<td><strong>contains(Object element)</strong></td>
</tr>
<tr>
<td>Returns true if this ArrayList instance contains the specified element.</td>
</tr>
<tr>
<td><strong>ensureCapacity(int minCapacity)</strong></td>
</tr>
<tr>
<td>Increases the capacity of this ArrayList instance, if necessary, to</td>
</tr>
<tr>
<td>ensure that it can hold at least the number of elements specified by</td>
</tr>
<tr>
<td>the minimum capacity argument.</td>
</tr>
<tr>
<td><strong>get(int index)</strong></td>
</tr>
<tr>
<td>Returns the element at the specified position in this list.</td>
</tr>
<tr>
<td><strong>indexOf(Object element)</strong></td>
</tr>
<tr>
<td>Searches for the first occurrence of the given argument, testing for</td>
</tr>
<tr>
<td>equality using the equals method.</td>
</tr>
<tr>
<td><strong>isEmpty()</strong></td>
</tr>
<tr>
<td>Tests if this list has no elements.</td>
</tr>
<tr>
<td><strong>lastIndexOf(Object element)</strong></td>
</tr>
<tr>
<td>Returns the index of the last occurrence of the specified object in</td>
</tr>
<tr>
<td>this list.</td>
</tr>
<tr>
<td><strong>remove(int index)</strong></td>
</tr>
<tr>
<td>Removes the element at the specified position in this list.</td>
</tr>
<tr>
<td><strong>removeRange(int fromIndex, int toIndex)</strong></td>
</tr>
<tr>
<td>Removes from this list all of the elements whose index is between</td>
</tr>
<tr>
<td>fromIndex, inclusive and toIndex, exclusive.</td>
</tr>
<tr>
<td><strong>set(int index, Object element)</strong></td>
</tr>
<tr>
<td>Replaces the element at the specified position in this list with the</td>
</tr>
<tr>
<td>specified element.</td>
</tr>
<tr>
<td><strong>size()</strong></td>
</tr>
<tr>
<td>Returns the number of elements in this list.</td>
</tr>
<tr>
<td><strong>toArray()</strong></td>
</tr>
<tr>
<td>Returns an array containing all of the elements in this list in the</td>
</tr>
<tr>
<td>correct order.</td>
</tr>
<tr>
<td><strong>toArray(Object[] a)</strong></td>
</tr>
<tr>
<td>Returns an array containing all of the elements in this list in the</td>
</tr>
<tr>
<td>correct order; the runtime type of the returned array is that of the</td>
</tr>
<tr>
<td>specified array.</td>
</tr>
<tr>
<td><strong>trimToSize()</strong></td>
</tr>
<tr>
<td>Trims the capacity of this ArrayList instance to be the list's current</td>
</tr>
<tr>
<td>size.</td>
</tr>
</tbody>
</table>
MinimalArrayList, THE REDUCED List CLASS

- 12 methods
- that's better!

- add(Object o)
- clear()
- contains(Object o)
- get(int index)
- indexOf(Object o)
- isEmpty()
- iterator()
- ListIterator(int index)
- remove(int index)
- remove(Object o)
- set(int index, Object o)
- size()
public class Cell {
    public String value;
    public Cell link;

    public Cell(String s) {
        value = s;
        link = null;
    }

    public void linkTo(Cell c) { link = c; }

    public void printAll() {
        System.out.print(value);
        if (this.link != null)
            link.printAll();
    }
}

LINKING Cell OBJECTS TO CREATE A ‘LIST’ (JAVA)
public class CellDemo {
    public static void main(String[] args) {
        Cell c1 = new Cell("1");
        Cell c2 = new Cell("2");
        Cell c3 = new Cell("3");
        Cell c4 = new Cell("End");

        // link them together from c1 to c4:
        c1.linkTo(c2);  // or c1.link = c2;
        c2.linkTo(c3);
        c3.linkTo(c4);

        // or build a linked list, one node at a time,
        // adding to the front of the list.

        Cell front = new Cell("End");
        for (int i = 0; i < 10; i++) {
            Cell newFront = new Cell("" + i);
            newFront.linkTo(front);
            front = newFront;
        }
        front.printAll();
    }
}

A DRIVER TO LINK UP Cell INSTANCES INTO A LIST (JAVA)
AND NOW A PYTHON IMPLEMENTATION OF A ‘LIST’

```python
class Cell:
    def __init__(self, value):
        self.value = value
        self.link = None

    def linkTo(self, cell):
        self.link = cell

    def getLink(self):
        return self.link

    def setValue(self, value):
        self.value = value

    def getValue(self):
        return self.value
```

AND NOW A PYTHON IMPLEMENTATION OF A ‘LIST’
c1 = Cell(1)
c2 = Cell(2)
c3 = Cell(3)
c4 = Cell(4)

c1.linkTo(c2)
c2.linkTo(c3)
c3.linkTo(c4)

def printValues(cell):
    while cell is not None:
        print cell.value
        cell = cell.link

printValues(c1)

1
2
3
4
HOW TO SUM UP THE VALUES IN THE LIST?

want to use iteration? or recursion?

def sum_iterative(cell):
    v = 0
    while cell is not None:  # or simply while cell:
        v += cell.value
        cell = cell.link
    return v

def sum_recursive(cell):
    if cell.link == None:
        return cell.value
    else:
        return cell.value + sum_recursive(cell.link)

or how about just asking the list to sum itself up and return the answer?
def printAll(self):
    print self.value
    if self.link is not None:
        self.link.printAll()

def sum_v1(self, cell, v):
    if cell.link == None:
        return cell.value + v
    else:
        return self.sum_v1(cell.link, v + cell.value)

def sum_v2(self, v):
    if self.link == None:
        return self.value + v
    else:
        return self.link.sum_v2(v + self.value)

def sum_v3(self):
    if self.link is None:
        return self.value
    else:
        return self.value + self.link.sum_v3()
def sum(self):
    if self.link is None:
        return self.value
    else:
        return self.value + self.link.sum()

or

def sum(self):
    if self.link:
        return self.value + self.link.sum()
    else:
        return self.value

AND FINALLY, AN ELEGANT RECURSIVE SUMMATION METHOD
HOW ABOUT USING A VISITOR TO SUM UP THE VALUES?

• add to Cell the following:
  ```python
def acceptVisitor(self, v):
    v.visit(self)
  ```

• create a Visitor hierarchy:
  ```python
class Visitor(object):
    def visit(self, cell):
      pass

class SumVisitor(Visitor):
  def __init__(self):
    self.sum = 0

  def visit(self, cell):
    self.sum += cell.value
    if cell.link is not None:
      cell.link.acceptVisitor(self)
  ```

• use it:
  ```python
sv = SumVisitor()
c1.acceptVisitor(sv)
print sv.sum
```
NOW, SHOULD Cell BECOME A “LIST” CLASS

- a Cell is just a modest little thing (a value and a reference to the next cell)
- it is thus a recursive data type
- should it also be able to do recursion on itself to perform:

  - add(o)
  - clear()
  - contains(o)
  - get(index)
  - indexOf(o)
  - isEmpty()
  - iterator()
  - ListIterator(index)
  - remove(index)
  - remove(o)
  - set(index, o)
  - size()
Is a Collection itself a Collection (i.e., recursive)?

- recall we asked, what sort of object is:
  - a bag of rocks?
  - a horde of monsters?
  - etc.

- so what sort of object is a Collection?
- is a Cell a Collection?
- in Java, an ArrayList ...
  - is an AbstractList ...
    - which is an AbstractCollection ...
      - which is a Collection
  - an ArrayList also implements List
- and what about a Cell?
  - can we (and should we) make a Cell implement List?
- should a Collection be a recursive data type, or should it just contain one?
if a Collection class were just a wrapper

- it could contain a (private) recursive data type (e.g., a list of linked `Cell` instances)
- it could then return an Iterator etc., but not itself be recursive
- that might finesse some efficiency issues
  - of adding at the head versus the tail of a list
  - of needing to shift all the elements of an ArrayList when adding to the zeroth element
A LIST CLASS
AN EXAMPLE OF A RECURSIVE DATA TYPE

• if we don’t want to just list any object, provide an interface for things that can be listed:

```java
interface Item {}
```

• provide a minimum set of requirements for any List implementation:

```java
interface ListI {
    public int getLength();
    public void setItem(Item c);
    public void setNext(List l);
    public Item getItem();
    public List getNext();
}
```
a List is usually either null (= the empty list), or a "node" containing a reference to an Item plus a next link

```java
import ListI;

class List implements ListI {
    Item item;
    List next;

    public List(Item c, List l) {
        setItem(c);
        setNext(l);
    }

    public int getLength() {
        return (next == null) ? 1 : next.getLength() + 1;
    }

    public void setItem(Item c) { item = c; }
    public void setNext(List l) { next = l; }
    public Item getItem() { return item; }
    public List getNext() { return next; }
}
```
AN OBJECT-ORIENTED LIST

- using the same interface

```java
interface ListI {
    public int getLength();
    public void setItem(Item c);
    public void setNext(List l);
    public Item getItem();
    public List getNext();
}
```

- let's make an abstract class to represent the recursive definition of a list

```java
abstract class List implements ListI {}
```

- and start the recursive definition of a list with a class representing the empty list, which has no length, whose value is null, and whose next is the empty list

```java
class EmptyList extends List {
    public int getLength() { return 0; }
    public void setItem(Item c) {}
    public void setNext(List l) {}
    public Item getItem() { return null; }
    public List getNext() { return this; }
}
```
and now we complete the recursive definition of a list with a class representing the non-empty list, which is a Node which references an Item and a List

```java
class Node extends List {
    Item item;
    List next;

    public Node(Item c, List l) {
        setItem(c);
        setNext(l);
    }

    public void setItem(Item c) { Item = c; }
    public void setNext(List l) { next = l; }
    public Item getItem()       { return Item; }
    public List getNext()       { return next; }
    public int  getLength()     { return next.getLength() + 1;  }
}
```
TOWARDS AN INDUSTRIAL-STRENGTH LIST CLASS

- interface Item {}

  class EmptyListE extends Exception {}

interface ListI {
  public void setItem(Item c) throws EmptyListE;
  public void setNext(List l) throws EmptyListE;
  public Item getItem() throws EmptyListE;
  public List getNext() throws EmptyListE;
  public int getLength();
}
abstract class List implements ListI {
}

public class EmptyList extends List {

    public void setItem(Item i) throws EmptyListE {
        throw new EmptyListE();
    }

    public void setNext(List l) throws EmptyListE {
        throw new EmptyListE();
    }

    public Item getItem() throws EmptyListE {
        throw new EmptyListE();
    }

    public List getNext() throws EmptyListE {
        throw new EmptyListE();
    }

    public int getLength() { return 0; }
}
class Node extends List {
    Item item;
    List next;

    public Node(Item i, List l) {
        setItem(i);
        setNext(l);
    }

    public void setItem(Item i) throws EmptyListE { item = i; }
    public void setNext(List l) throws EmptyListE { next = l; }

    public Item getItem() throws EmptyListE { return item; }
    public List getNext() throws EmptyListE { return next; }
    public int getLength() { return next.getLength() + 1; }
}
class ListC implements CollectionI {
    private List list;

    public ListC() {
        list = new EmptyList();
        count = 0;
    }

    public Item getNextItem() throws CollectionE {
        Item item;

        try {
            item = ... fetch the next element
        }
        catch (EmptyListE e) {
            throw new CollectionE();
        }
        return item;
    }
}
much more OO Lists

- how to add to the tail of a list?
  - growing a list using the Decorator Pattern?
- how to detect breaks? (Observer Pattern?)
- how to detect cycles?
- for ordered lists (Visitor Pattern?)
  - how to add a new node at the correct position?
  - make a list self-sorting?
  - reverse order of list?
- how to introduce the State Pattern (Empty or Node)?
- Iteration by Visitor?
A STRATEGY-PATTERN LIST, MAYBE?

- using the same interface
  ```java
  interface ListI {
    public int getLength();
    public void setItem(Item c);
    public void setNext(List l);
    public Item getItem();
    public List getNext();
  }
  ```

- let's make an class to represent the recursive definition of a list
  ```java
  public class List implements ListI {
    protected int getLengthStrategy;
    protected int setItemStrategy;
    protected int getItemStrategy;
    protected int setNextStrategy;
    protected int getNextStrategy;
    ...
  }
  ```

- plus write 2 x 5 strategies and 2 subclasses (EmptyList and Node)
OR A STATE-PATTERN LIST, PERCHANCE?

- create an interface to describe all state-dependent methods

```java
interface ListState implements ListI {
    abstract public int getLength();
    abstract public void setItem(Item c);
    abstract public void setNext(List l);
    abstract public Item getItem();
    abstract public List getNext();
}
```

- let's make an class to represent the recursive definition of a list

```java
class List implements ListI {
    protected ListState currentState;
}
```

- plus write two ListState-implementing classes (EmptyState and NodeState)