Choices

- As we know, arrays are just one kind of list
  - Arrays are great for some problems, but not all
    - Things are not usually one-size-fits-all
    - What if your algorithm requires *lots* of insertion/deletion?
  - *We want options!*

- Maybe we can define other kinds of lists that have tradeoffs:
  - Some kinds of lists might have *faster* add() and remove() times
    - But maybe get() becomes slower
  - Other kinds of lists might be able to grow and shrink
    - But maybe that would cause insertion/deletion time to suffer
For each type of list, we need to model the time complexity of its common operations

- e.g., construction, adding, removing, searching, referencing

Understanding the complexity of its operations allows us to make the proper choice when programming

- Use the one that'll give you the best performance on average
- One of the most important skills
  - Need to have a fundamental understanding of data structures' (e.g., lists) implementation
Outline

- Java's List Interface
  - ArrayList
    - Aside: Generics
  - Linked List
- Iterator
- Conclusion
Java provides a `List<E>` interface

- The diamond notation `<E>` allows for "generic" objects to be stored
- Full API: [https://docs.oracle.com/javase/8/docs/api/java/util/List.html](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public E get(int index)</code></td>
<td>Returns the data in the element at position <code>index</code></td>
</tr>
<tr>
<td><code>public E set(int index, E item)</code></td>
<td>Stores a reference to <code>item</code> in the element at position <code>index</code>. Returns the data formerly at position <code>index</code></td>
</tr>
<tr>
<td><code>public int size()</code></td>
<td>Return the current size of the <code>List</code></td>
</tr>
<tr>
<td><code>public boolean add(E item)</code></td>
<td>Adds a reference to <code>item</code> at the end of the <code>List</code>. Always returns <code>true</code></td>
</tr>
<tr>
<td><code>public boolean add(int index, E item)</code></td>
<td>Adds a reference to <code>item</code>, inserting it before the item at position <code>index</code></td>
</tr>
<tr>
<td><code>public int indexOf(E item)</code></td>
<td>Searches for <code>target</code> and returns the position of the first occurrence, or <code>−1</code> if it is not in the <code>List</code></td>
</tr>
<tr>
<td><code>public E remove(int index)</code></td>
<td>Removes the entry formerly at position <code>index</code> and returns it</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Java's List Interface (Cont.)

- List hierarchy and implementing classes

```
<<interface>>
List

<<implements>>
AbstractList

<<extends>>
ArrayList
Vector
AbstractSequentialList

<<extends>>
Stack
LinkedList
```
Outline

- Java's List Interface
  - ArrayList
- Aside: Generics
- Linked List
- Iterator
- Conclusion
In This Lecture

- We will implement two important List classes
  - ArrayList and LinkedList
    - To avoid naming conflicts with Java, we name ours MyArrayList and SinglyLinkedList
  - They *ought to* implement Java's List interface, but we don't have time.
    - We'll roll our own MyList interface, which is just a subset of Java's.

- Why we're doing this:
  - To be exposed to various data structures
  - To understand performance tradeoffs between them
ArrayList Class (Review of CS 161)

- **Motivation**: Arrays cannot be resized easily
- Designers of Java understood this limitation, and created `ArrayList`
  - Can hold any number of objects
  - Can hold any type of object (generics)
  - Add and remove elements as needed (grow and shrink)
### Useful ArrayList Methods

Hey, these were all specified by the `List` interface!!!

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public boolean add(E e)</code></td>
<td>Inserts element e to end of this list</td>
</tr>
<tr>
<td><code>public boolean add(int index, E e)</code></td>
<td>Inserts element e at the specified position in the list</td>
</tr>
<tr>
<td><code>public void clear()</code></td>
<td>Removes all elements from list</td>
</tr>
<tr>
<td><code>public E get(int index)</code></td>
<td>Returns the element at given index</td>
</tr>
<tr>
<td><code>public int indexOf(E e)</code></td>
<td>Searches for e in the list, returns the index of the first occurrence if found, or -1 if not found.</td>
</tr>
<tr>
<td><code>public E remove(int index)</code></td>
<td>Removes the element at given index. Returns the deleted element</td>
</tr>
<tr>
<td><code>public E set(int index, E e)</code></td>
<td>Replaces the element at the specified index.</td>
</tr>
<tr>
<td><code>public int size()</code></td>
<td>Returns the number elements in the current list</td>
</tr>
<tr>
<td><code>public String toString()</code></td>
<td>Returns the String representation of the current list</td>
</tr>
</tbody>
</table>

For the full ArrayList API: [http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)
An ArrayList is implemented over an array

- Think of arrays as stadium/theater seating
  - Seat can't be moved (This is the biggest limitation)
  - Number of seats don't change once the row of seats is created

List<String> dwarfs = new ArrayList<>(); // assume an empty array of size 10 is created
Adding to the **tail** of the list

- O(1) time!
Removing from the *tail* of the list is also a breeze!

- O(1) time!

```java
List<String> dwarfs = new ArrayList<>();
dwarfs.remove(dwarfs.size() - 1);
dwarfs.remove(dwarfs.size() - 1);
dwarfs.remove(dwarfs.size() - 1);
dwarfs.remove(dwarfs.size() - 1);
dwarfs.remove(dwarfs.size() - 1);
dwarfs.remove(dwarfs.size() - 1);
```
Need Volunteers

What if we added to the *head* of the list?

- Seems like it would be more than O(1)

```java
List<String> dwarfs = new ArrayList<>();
dwarfs.add(0, "Bashful");
dwarfs.add(0, "Jumpy");
dwarfs.add(0, "Snorty");
dwarfs.add(0, "Grumpy");
dwarfs.add(0, "Sneezy");
```
Need Volunteers

What if we removed from the *head* of the list?

```java
List<String> dwarfs = new ArrayList<>();
dwarfs.remove(0);
dwarfs.remove(0);
dwarfs.remove(0);
dwarfs.remove(0);
dwarfs.remove(0);
dwarfs.remove(0);
```
What if we run out of room in our internal array when adding?

List<String> dwarfs = new ArrayList<>(); // assume an empty array of size 10 is created
dwarfs.add("Bashful");
dwarfs.add("Jumpy");
dwarfs.add("Snorty");
dwarfs.add("Grumpy");
dwarfs.add("Sneezy");
dwarfs.add("Doc");
dwarfs.add("Coughy");
dwarfs.add("Happy");
dwarfs.add("Hangry");
dwarfs.add("Careful"); // well, that's 10
dwarfs.add("Quincy"); // here's the 11th (obviously, ArrayLists allow for this)
Details of ArrayList

- Some details you can now appreciate:
  - Built on top of arrays
    - Unless otherwise specified when constructing, has an initial capacity of 10
  - Automatic reallocation
    - Can add 11th element, but not before array size doubles automatically!
    - In fact, will always double capacity when adding new element exceeds capacity
The MyList Interface

- Here's our pared down version of Java's List interface
  - Assume MyLists only hold doubles for now.

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean add(double item)</td>
<td>Adds item at the end of the list. Always returns true</td>
</tr>
<tr>
<td>public boolean add(int index, double item)</td>
<td>Adds a reference to item, inserting it before the item at position index</td>
</tr>
<tr>
<td>public E get(int index)</td>
<td>Returns the data in the element at position index</td>
</tr>
<tr>
<td>public int indexOf(double item)</td>
<td>Searches for target and returns the position of the first occurrence, or –1 if it is not in the List</td>
</tr>
<tr>
<td>public double remove(int index)</td>
<td>Removes the entry formerly at position index and returns it</td>
</tr>
<tr>
<td>public E set(int index, double item)</td>
<td>Stores a reference to item in the element at position index. Returns the data formerly at position index</td>
</tr>
<tr>
<td>public int size()</td>
<td>Return the current size</td>
</tr>
</tbody>
</table>

- Let's write our own MyArrayList to see how it's really implemented!
  - (Code on course page)