Building an extensible container class in Java.

1. Questions?

2. A quick note about the *type hierarchy* in Java.
   
   (a) Every new type of object is a *subtype* of some older type (its *supertype*)
   
   (b) The ultimate supertype of every object in Java is an *Object*
   
   (c) Type extension tells us that if a piece of code performs a calculation based on a specific type, say an *Automobile*, any object of a more specific subtype, like *Tesla* will work in the calculation as well.
   
   (d) It’s always legal to assign a subtype value to a supertype variable:

   ```java
   Automobile a = new Tesla();
   ```
   
   (e) The other way is not advisable. Not all *Automobiles* have a `charge()` method, so the assignment of a supertype (*Automobile*) to subtype variable (*a Tesla*) is generally illegal.

   (f) On occasion, we *know* the assignment is legal. We can inform Java of this fact by specifying or *casting* the object’s known type. Consider the following:

   ```java
   Automobile a = new Tesla();
   a.driveTo("Toronto");
   Tesla t = (Tesla)a; //this is a case: we *know* a is a Tesla
   t.charge();
   ```

3. Implementing a *List-like class*. Based on *arrays* of *Objects*.

   (a) An array can be allocated with a specific length. Here, we allocate an array with 10 elements.

   ```java
   Object[] data = new Object[10];
   ```

   Where `n` is the number of cells required. Note that this allocates an array of `n` *Object references*; there is no need to think about how they are constructed. Notice that an array is allocated with a specific length. We can determine the length of the array with the `length` instance variable. The length cannot be changed. That’s our motivation for building the *List* class.

4. What would we need to keep track of, privately, to maintain state? An approach: keep an array that is *at least as long* as the desired List. We must do the bookkeeping associated with knowing whether we need to expand the List. How?

5. What would be an appropriate constructor? For each constructor, we should be able to determine reasonable values of the state variables; the constructor should leave the list in a valid state.
6. How do we implement int size()? This method is the equivalent of __len__ in python. Remember to distinguish a list's logical size from its physical capacity.

7. How do we add a value to the list? This should be most like list.append(o) in Python. How do we determine if a List has the ability to hold n values? Thinking about how this is implemented is the most important part of the engineering of this structure.

8. How do we implement Object get(int n)? This is most like list's __getitem__ method. How does the user treat the return value? Do we have similar problems with void set(int i, Object v)?

9. How do we implement void insert(int i, Object v)? Where should it be added? What is the cost of implementing this method?

10. It should, then be straight forward to implement boolean contains(Object v). This method is similar to __contains__ in python's list class. The python method is invoked when you use the in keyword. In Java, there is no such shortcut.

*