Lists and Iterators
Arrays Review

- Arrays are a simple data structure

- Arrays store a row of values of the same type
  - Primitive types (int, double, etc.)
    ```java
    // array that can hold 10 chars
    char[] letters = new char[10];
    ```
  - Objects (Students, Dates etc.)
    ```java
    // array that can hold 3 LightSwitches
    LightSwitch[] switches = new LightSwitch[3];
    ```
Arrays Review

- **Access each value through an index:**
  
  ```java
  int[] intArray = new int[20];
  intArray[0] = 4; intArray[1] = 75;
  ```

- **Array indices start at 0, not 1**
  - first element of `intArray` is `intArray[0]`
  - last element of `intArray` is `intArray[19]`

- **Important**: Array lengths cannot be changed once they are declared!
Is there something better?

- As we learned in the *Gradebook* lab, because of their fixed length, arrays can be annoying to use.

- Is there something like an array but that will handle all the resizing automatically?

- YES!
**ArrayList**

- **ArrayList** stores its elements internally as an array.
  - `ArrayList newList = new ArrayList();`

- **get** method is *fast* – just retrieves the element at the specified index

- **add** method is *slow* – may have to create a larger array and copy over all the elements.
A linked list is like a freight train: each link stores an item and is connected to the next link.

The list holds a reference to the first (and maybe the last) element.
**LinkedList**

- LinkedList stores its elements internally in a linked list data structure.
  - LinkedList anotherList = new LinkedList();

- add method is *fast* – just appends a new link to the end of the list

- get method is *slow* – has to walk down the list retrieve the element at the specified index
iterator method

- Both `ArrayList` and `LinkedList` have a method named `iterator`.

- Returns an object of type `Iterator`.

- We use the `Iterator` to iterate over the elements of the list.
Iterators

- `hasNext` method returns true when there are more elements to iterate over.

- `next` method returns the next element in the iteration.
Because the next method returns type `Object`, you must cast the return value to the actual type of the value.

"Casting" means "promising" the compiler that the object will be of a particular type.

This allows you to use methods of the actual type without the compiler complaining.

Example on next slide . . .
class GradeBook {
    private ArrayList grades;

    void printGrades() {
        Iterator gradeIter = grades.iterator();
        while (gradeIter.hasNext()) {
            Double g = (Double) gradeIter.next();
            System.out.println(g.doubleValue());
        }
    }
}
Quiz

- Which list implementation is fast when accessing arbitrary indices? **ArrayList**

- What is wrong with the iteration below? **Casting**

```java
for(Iterator studentIter = students.iterator();
    studentIter.hasNext();)
    { 
    Student s = (Student)studentIter.next();
    System.out.println(s.getDescription());
}
```