CIS 211
Lecture 2

ArrayList
reading: 10.1

based on slides created by Marty Stepp
http://www.cs.washington.edu/143/

What’s wrong with this code?

// Code from CIS 210 Assignment 6, by Michal Young
class MuppetMaster {
    static final String infile = "costumes.txt";
    static final int MAXMUPPETS = 100;
    Muppet[] muppets = new Muppet[MAXMUPPETS];
    int muppetCount = 0;

    // Instantiates the class and start its main instance method.
    public static void main(String[] args) throws FileNotFoundException {
        // Create a scanner to read the input file
        Scanner in = new Scanner(new File(infile));
        MuppetMaster me = new MuppetMaster();
        me.analyzeCostumes(in);
        // The input has been read, now provide output
        me.printMuppetStats();
        // ... Etc. ...
    }
}

Another example...

/**
 * Represent a human hand.
 * @author Inigo Montoya
 */

class Hand {
    Finger[] fingers = new Finger[5];
    boolean isRightHand = false;

    // ...
}

Zero, One, Infinity (ZOI)

Rule-of-thumb in software engineering:
There are only three numbers – zero, one, and infinity

In other words, don’t place arbitrary limits on the number of entities.

"Two is an impossible number, and can’t exist."
Isaac Asimov, The Gods Themselves

Exhibit A: A flawed class.
(Count Rugen, from “The Princess Bride.”)

Exhibit B: A six-fingered man.
One solution...

```java
/**
 * Represent a human hand.
 * @author Inigo Montoya
 */
class Hand {
    Finger[] fingers;
    boolean isRightHand = false;

    /**
     * Construct hand with specific number of fingers.
     */
    Hand(int numFingers) {
        if (numFingers == 6) {
            System.out.println("You killed my father.");
            System.out.println("Prepare to die!");
        }
        fingers = new Finger[numFingers];
    }
}
```

What about Muppets?

In the muppet program, we need to read in information about an unknown (but possibly large) number of muppets.

Will such an approach work there?

Why or why not?

Goal

We want to maintain a list of objects that will automatically grow as we add to it, e.g.:

- List of students in a class (Blackboard)
- List of events in a calendar (Google Calendar, iCal)
- List of emails in an inbox (Any email program)

Lists

- **list**: a collection storing an ordered sequence of elements
  - each element is accessible by a 0-based **index**
  - a list has a **size** (number of elements that have been added)
  - elements can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **ArrayList** object
Idea of a list

• Rather than creating an array of boxes, create an object that represents a “list” of items. (Initially an empty list.)

  []

• You can add items to the list.
  – The default behavior is to add to the end of the list.
    [hello, ABC, goodbye, okay]

• The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  – Think of an “array list” as an automatically resizing array object.
  – Internally, the list is implemented using an array and a size field.

ArrayList methods (10.1)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>appends value at end of list</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td>indexOf(value)</td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>get(index)</td>
<td>returns the value at given index</td>
</tr>
<tr>
<td>remove(index)</td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td>set(index, value)</td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>toString()</td>
<td>returns a string representation of the list such as “[3, 42, -7, 15]”</td>
</tr>
</tbody>
</table>

ArrayList methods 2

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addAll(list)</td>
<td>adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)</td>
</tr>
<tr>
<td>addAll(index, list)</td>
<td>adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)</td>
</tr>
<tr>
<td>contains(value)</td>
<td>returns true if given value is found somewhere in this list</td>
</tr>
<tr>
<td>containsAll(list)</td>
<td>returns true if this list contains every element from given list</td>
</tr>
<tr>
<td>equals(list)</td>
<td>returns true if given other list contains the same elements</td>
</tr>
<tr>
<td>iterator()</td>
<td>returns an object used to examine the contents of the list (seen later)</td>
</tr>
<tr>
<td>listIterator()</td>
<td>returns an object used to examine the contents of the list (seen later)</td>
</tr>
<tr>
<td>lastIndexOf(value)</td>
<td>returns last index value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>remove(value)</td>
<td>finds and removes the given value from this list</td>
</tr>
<tr>
<td>removeAll(list)</td>
<td>removes any elements found in the given list from this list</td>
</tr>
<tr>
<td>retainAll(list)</td>
<td>removes any elements not found in given list from this list</td>
</tr>
<tr>
<td>subList(from, to)</td>
<td>returns the sub-portion of the list between indexes from (inclusive) and to (exclusive)</td>
</tr>
<tr>
<td>toArray()</td>
<td>returns the elements in this list as an array</td>
</tr>
</tbody>
</table>

Sneak Peek

• An ArrayList can be a collection of any type of object. (Just like an array.)
• Next week, we’ll learn how to use them and other collections.
• This week, we’re going to build a special case of ArrayList from the ground up:

  ArrayIntList

  (Chapter 15.1, 15.2)