CSE 143
Computer Programming II
ArrayIntList

I made a JavaList!

[Espresso, cappuccino, mocha]
Questions From Last Time

- Can you cover which guidelines are being used for style?
- What is the type of an ArrayList that has 3.5, 2.7, etc. (ArrayList<Double>)
- I took CSE 142 (or equivalent) a long time ago. What should I do? What do I need to know from 142?
- This is a test to see if you actually read all of the notecards.
- Do different companies have different style guidelines? (Yes.)
- What defines a “good comment”? (length? can they be too long?)
- What are design decisions? (See Piazza)
Questions From Last Time

- Coming to class late? Leaving early?
- Are exams open note? (No, but we give you a cheatsheet attached to the exam)
- Is the class curved?
- Lighter color on slides?
- JGrasp vs. Eclipse?
- Is there a style guide?
- Is there such a thing as too many comments?
Drawings

This is the happy screen!

These are his friends

Yay!

C <<< >
from the balcony you only see your top half
so I assume you're a sock puppet.

How Rainbows are made
Wrapper Classes

int vs. Integer      char vs. Character    double vs. Double

The **lowercase** versions are **primitive types**; the **uppercase** versions are “wrapper classes”.

The following is valid code:

```java
1  int a = 5;
2  Integer b = 10;
3  int c = a + b;  // You can treat ints and Integers as the same
```

When we create ArrayList’s, we must use **non-primitive types**. So:

```java
1  ArrayList<int> bad1 = new ArrayList<int>();  // This won’t compile!
2  // v This will work.
3  ArrayList<Integer> better = new ArrayList<Integer>();
4  better.add(5);  // We can add an ’int’ to an ’Integer’ ArrayList
```
Clients and Implementors

Client vs. Implementor: Medication

For a tylenol pill, who is the client? Who is the implementor?

Java Examples

You’ve already been a client!

- DrawingPanel
- ArrayList

You’ve already been an implementor!

- Critter
Class

A **Class** is

- a complete program, or
- a “template” for a type

(Examples: ArrayList, ReverseFile, ...)

The class explains what an object is, an **instance** is a particular version of the object.

```java
1 ArrayList<String> list1 = new ArrayList<String>();
2 ArrayList<String> list2 = new ArrayList<String>();
3 //list1 and list2 are instances of ArrayList
```

Object

An **Object** combines **state** and **behavior**.

Java is an “object-oriented” programming language (OOP); programs consist of objects interacting with each other.
A class is made up of **field(s)**, **constructor(s)**, and **method(s)**. Let's make an object `Circle` that represents a circle...

- with a size
- that can be moved right
- at a particular location

```java
public class Circle {

    /* Fields */
    private int radius;
    private int x;
    private int y;

    /* Constructor */
    public Circle(int radius, int x, int y) {
        this.radius = radius;
        this.x = x;
        this.y = y;
    }

    /* Methods */
    public void moveRight(int numberOfUnits) {
        this.x += numberOfUnits;
    }
}
```
**Implementor View of ArrayList**

What behavior should we support? (Methods)
- add, remove, indexOf, etc.

What state do we keep track of? (Fields)
- Elements stored in the ArrayList (probably stored as an array!)
- Size of ArrayList

**Two Views of an ArrayList**

<table>
<thead>
<tr>
<th>Client View:</th>
<th>3</th>
<th>-23</th>
<th>-5</th>
<th>222</th>
<th>35</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Impl. View:</th>
<th>3</th>
<th>-23</th>
<th>-5</th>
<th>222</th>
<th>35</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
- No generics (only stores ints)
- Fewer methods: add(value), add(index, value), get(index), set(index, value), size(), isEmpty(), remove(index), indexOf(value), contains(value), toString()
# Implementing `add`

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>2</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

`lst.add(222):`

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>2</td>
<td>45</td>
<td>222</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

How do we add to the end of the list?

- Put the element in the last slot
- Increment the size

```java
public void add(int value) {
    this.data[size] = value;
    size++;
}
```
System.out.println automatically calls toString on the given object. toString looks like:

1 public String toString() {
2     ... 
3 } 

ArrayIntList toString:

1 public String toString() {
2     if (size == 0) {
3         return "[]"; 
4     }
5     else {
6         String result = "[" + this.data[0];
7         for (int i = 1; i < this.size; i++) {
8             result += ", " + this.data[i];
9         }
10         result += "]";
11         return result;
12     }
13 }
Implementing add #2

```
public void add(int index, int value) {
    for (int i = size; i > index; i--) {
        this.data[i] = this.data[i - 1];
    }
    this.data[index] = value;
    size++;
}
```

How do we add to the middle of the list?

- Shift over all elements starting from the end
- Put the new element in its index
- Increment the size
Today’s Takeaways!

- Understand the difference between client and implementor

- Always use wrapper classes when creating an ArrayList of a primitive type

- Understand how ArrayList is implemented