More ArrayIntList; pre/post; exceptions; debugging
Questions Part 1

- Eclipse is not installing: (1) make sure you are downloading Neon, (2) uninstall and reinstall the JDK, (3) come to office hours/IPL.
- Is Viz open source? Not yet, but it will be.
- Integer.toString(i) or i + ""; either is fine.
- You should always use "this" when you can—see the style guidelines.
- Can we use Javadoc (e.g., "@params")? Yes.
- Are there any bugs that have shipped in Java. Yes; we'll talk about some later.
- Do I have to put my curly brace on the same line? No, but be consistent.

Questions Part 2

- What's the difference between primitives and Objects? Primitives are built into Java and handled differently internally. You really don't have to know the difference, but I can explain in depth at office hours.
- Mac or PC? Doesn't matter!
- Use Java 1.8
- How do I do well in this class? Do LOTS of practice-it problems.
- When will we have everything we need for homework? It varies week-by-week. You can almost always start the HW the day it comes out; we may go through more useful things throughout the week, but they shouldn't block you from making progress.

Where Are We Again?

- What Are We Doing...?
  We're implementing our own (simpler) version of ArrayList to (a) see how it works, and (b) get experience being the "implementor" of a class.

  Today's Main Goal:
  To finish ArrayIntList!

Outline

1. Debugging
2. Removing Code Duplication
3. Improving Readability!
4. Preventing Malicious Behavior
5. Re-structuring the Code
6. Extras
WTF’s per Minute

**Code Quality Measurement:**

**WTFs/Minute**

![WTFs/Minute](http://compadre.com)

Good Code

- Good
- Good
- Good

Bad Code

- Bad
- Bad
- Bad

NullPointerException

In addition, we run the following line:

```java
this.elementData[size] = element;
```

By default, unless we use the new keyword, all arrays are null. null is Java’s way of saying “you haven’t told me what to put here yet”. So, if `this.elementData` is null, then what is `null[size]`?

The takeaway here is that finding an “index” of null doesn’t make sense. We should ensure that we assign `this.elementData` to be a new array if we want to use it.

Duplicated Code: Methods

**Redundant add Methods**

1 /* Inside the ArrayIntList class... */
2 public void add(int value) {
3     this.set(size, value); /* THIS LINE IS DUPLICATED BELOW!!! */
4     size++; /* THIS LINE IS DUPLICATED BELOW!!! */
5 }
6
7 // Inserts value into the list at index.
8 // public void add(int index, int value) {
9     for (int i = size; i > index; i--) {
10         this.set(i, this.get(i-1));
11     }
12     this.set(size, value); /* THIS LINE IS DUPLICATED ABOVE!!! */
13     size++; /* THIS LINE IS DUPLICATED ABOVE!!! */

The fix is to call the more general add method from the less general one. (As a rule of thumb, methods with fewer arguments are less general.) So, we’d replace the first method with:

**Fixed add Method**

1 public void add(int value) {
2     add(this.size, value);
3 }

Rubber Ducky, You’re The One!

**A Bug?**

```java
public String toString() {
    String result = "[";
    for (int i = 0; i < this.size() - 1; i++) {
        result += this.elementData[i] + ", ";
    }
    result += this.elementData[this.size() - 1];
    result += "]";
    return result;
}
```

Rubber Duck Debugging

Rubber Ducky, You’re The One! is the idea that when your code doesn’t work, you talk to an inanimate object about what it does to find the error.

The idea is to verbalize what your code is supposed to do vs. what it is doing. Just saying it out loud helps solve the problem.

Hitting Capacity

**Example ArrayList**

Client View: 29 1 3 9 8
Impl. View: 29 1 3 9 8

Let’s run `add(3, 8)`! Uh oh! There’s no space left. What do we do?

Create a new array of `double` the size, and copy the elements!

**Resizing (Implementor View)**

Before: 29 1 3 9 8
Resize: 29 1 3 9 8 0 0 0 0
Insert: 29 1 3 8 9 8 0 0 0 0

Duplicated Code: Constructors

We’d like to have two constructors for ArrayIntList:

- One that uses a default size
- One that uses a size given by the user

**Redundant Constructors**

```java
public ArrayIntList() {
    this.data = new int[10];
    this.size = 0;
}
```

This is a lot of redundant code! How can we fix it?

**Fixed Constructor**

Java allows us to call one constructor from another using `this(...)`: 1 public ArrayIntList() {
2     this(10);
3 }

WTF’s per Minute

**Code Quality Measurement:**

**WTFs/Minute**

![WTFs/Minute](http://compadre.com)

Good Code

- Good
- Good
- Good

Bad Code

- Bad
- Bad
- Bad
Looking back at the constructor, what’s ugly about it?

```
1 public ArrayIntList() {
2   this(10);
3 }
```

The 10 is a “magic constant”; this is really bad style! We can use:

```
public static final type name = value
```

to declare a class constant.

So, for instance:

```
public static final int DEFAULT_CAPACITY = 10.
```

### Class CONSTANTS
A class constant is a global, unchangable value in a class. Some examples:

- Math.PI
- Integer.MAX_VALUE, Integer.MIN_VALUE
- Color.GREEN
- 

### Preconditions
A precondition is an assertion that something must be true for a method to work correctly. The objective is to tell clients about invalid ways to use your method.

Example Preconditions:

- For moveRight(int numberOfUnits):
  ```java
  // pre: numberOfUnits >= 0
  ```
- For minElement(int[] array):
  ```java
  // pre: array.length >= 0
  ```
- For add(int index, int value):
  ```java
  // pre: capacity >= size + 1; 0 <= index <= size
  ```

Preconditions are important, because they explain method behavior to the client, but they aren't enough! The client can still use the method in invalid ways!

### Why Use Exceptions?
Exceptions prevent the client from accidentally using the method in a way it wasn’t intended. They alert them about errors in their code!

An Example

```
1 public void set(int index, int value) {
2   if (index < 0 || index >= size) {
3     throw new IndexOutOfBoundsException(index);
4   }
5   this.data[index] = value;
6 }
7
8 public int get(int index) {
9   if (index < 0 || index >= size) {
10      throw new IndexOutOfBoundsException(index);
11   }
12   return data[index];
13 }
```

Uh oh! We have MORE redundant code!
**Postconditions**

A **postcondition** is an assertion that something must be true after a method has run. The objective is to tell clients what your method does.

Example Postconditions:
- For `moveRight(int numberOfUnits)`:
  ```java
  // post: Increases the x coordinate of the circle by numberOfUnits.
  ```
- For `minElement(int[] array)`:
  ```java
  // post: returns the smallest element in array.
  ```
- For `add(int index, int value)`:
  ```java
  // post: Inserts value at index in the ArrayList; shifts all elements from index to the end forward one index; ensures capacity of ArrayList is large enough.
  ```

Postconditions are important, because they explain method behavior to the client.

**Using Fields Directly vs. Using Instance Methods**

```java
public class Circle {
  int radius;
  int x, y;

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

  public void setX(int x) {
    this.x = x;
  }

  public int getX() {
    return this.x;
  }

  /* There are two possible implementations of moveRight... */
  public void moveRight(int numberOfUnits) {
    this.x += numberOfUnits;
  }
  public void moveRight(int numberOfUnits) {
    this.setX(this.getX() + numberOfUnits);
  }
}
```

**Why Use Fields vs. Instance Methods?**

- **Why Use Fields Directly?**
  - It’s sometimes more readable to use the fields directly
  - The code is sometimes shorter

- **Why Use Getters and Setters?**
  - What happens if we change the implementation (e.g. Point location instead of int x, y)?
  - If there is code that checks validity of inputs, then we only put it in one place

**Implementing remove**

```java
(size = 5) 3 8 2 45 6 0 0 0
```

```java
list.remove(2):
(size = 3) 3 8 45 6 0 0 0 0
```

How do we remove from the middle of the list?
- Shift over all elements starting from the index to remove at
- Set the last element to 0 (Do we need to do this?)
- Decrement the size

And how does the client see all of our comments...?