Lecture 2: ArrayIntList design

- (ADD CONSTRUCTOR CALLS)
  - Use debugger to show what happens - each has its own elementData and size
  - But elementData is set to NULL - special 0-equivalent meaning "no value"
    - We never told it to construct a new array!
  - We could initialize it when declared, but this is BAD
    - Job of the constructor, a special method called when you create a new object using **new**
  - Constructor has special syntax - no return type, same name as the class
    - This is how Java knows to call it when you say **new**
  - ADD A CONSTRUCTOR
    - But we didn't have one before - why could we create one before?
    - Because Java provides you a default constructor if you don't specify one
    - If you provide a constructor, then Java assumes you know what you are doing
    - Initializes size and elementData
      - Better style to initialize fields in constructor - will LOSE POINTS
    - But doesn't technically need to initialize size, since already 0 - up to you
  - Rerun debugger - now elementData is initialized to size 100
- Adding to the list
  - Could directly manipulate elementData, size

```
list1.elementData[0] = 1;
list1.elementData[1] = 82;
list1.elementData[2] = 97;
list1.size = 3;
```

  - **What's the problem?**
  - Not very "object-oriented approach"
    - Don't want client to deal with the nitty-gritty details
    - What if you had to dig inside the transistors of the radio to change the channel?
  - Instead, the ArrayIntList itself should know how to add a value to it
  - Have an **add method** called from the client code - each time indicate which value to add
    - CHANGE THE CLIENT CODE
  - Takes a parameter that is the value to add, and will add to the end of the list
- How do we figure out where to add to the list?

```
list contents    size    where to add
-------------------------------------
[]                0      elementData[0]
[1]               1      elementData[1]
[1, 82]           2      elementData[2]
[1, 82, 97]       3
```

- Size --> where to add, and then size goes up by one

```
elementData[size] = value;
size++;
```

- What parameters does the add method need? Obviously the value
  - But how does the method get access to size/elementData?
    - An object knows its own fields
  - We're saying "Hey list1, I'm talking to you! Execute your add method with value of 1!)
    - **"Implicit parameter"** - modify fields of the object you're talking to.

- Great! Now let's print out the lists (ADD PRINTLN)
  - Doesn't give us much information
  - Java also automatically gives you a toString method (just like a constructor) but it's not very good.
  - Let's write our own
  - Print out values, comma-separated, between brackets

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

- I want to talk about encapsulation - back to the radio
  - Normally electronics have a plastic/metal case
  - The innards are encapsulated - cannot be seen or manipulated
  - On the back usually some kind of plate with screws with message "Do not remove, warranty void if removed"
  - Why the warning?
    - User could break something
  - Something similar can happen with our class
    - What is it?

```
list1.size = 10000;
list2.size = -384;
```

  - How to prevent it?
    - MAKE FIELDS PRIVATE
    - RERUN - now won't compile
    - This is VERY important - style points!
  - But client probably still wants to know the size
    - ADD SIZE METHOD - getter. This is typical
    - PRINT the size in client
    - aka "What station is the radio set to?"

- Other methods that the client might want to look at the list?
  - GET
  - Are there any problems with this?
    - Pre/post conditions - the contract
    - Assumptions that are made in order for the method to work
    - If client violates them, anything could happen
    - You MUST comment these in your code
  - If we want to do more, we can throw an exception
    - IndexOutOfBoundsException (must be COMMENTED)
    - We create a new Exception object and THROW it - stops execution
    - Exception object contains information about what happened when it was created
      - The info you see when you look in your console at runtime
    - Can also pass a String with more info to the Exception
  - Sometimes we create a special method just for the exception check
- indexOf

```
for (int i = 0; i < size; i++) {
   if (elementData[i] == value) {
       return i;
   }
}
return -1;
```

- contains
  - Not necessary, because client can use indexOf, but typical
  - What should the return type be?
  - BOOLEAN ZEN - write with an if/else, then say why it's bad (ch. 5)

```
return indexOf(value) != 0;
```

- isEmpty

```
 return size == 0;
```

- remove(index) - must shift everything over. Don't need to set last thing to 0.

```
checkIndex(index);
for (int i = index; i < size - 1; i++) {
   elementData[i] = elementData[i + 1];
}
size--;
```

- add(index, value) - must shift everything over (MUST go backwards)

```
if (index < 0 || index > size) {
   throw new IndexOutOfBoundsException("index: " + index);
}
if (size + 1 > elementData.length) {
   throw new IllegalStateException("would exceed list capacity");
}
for (int i = size; i > index; i--) {
   elementData[i] = elementData[i - 1];
}
elementData[index] = value;
size++;
```

- Back to the constructor
  - The size of 100 - what's the problem? Not very flexible
  - So instead, let's let the user tell us
    - Take a parameter in the constructor

```
public ArrayIntList(int capacity) {
    elementData = new int[capacity];
    size = 0;
}
```

    - CHANGE CLIENT
  - But what if the user doesn't know how big? ex. reading a file
    - We still want some kind of default
    - Try changing client code to remove the parameter!
    - But now we can't use the default constructor
    - Java provides you a default constructor, but only if you don't have any constructors
      - Assumes you know what you're doing
    - So add a default constructor

```
public ArrayIntList() {
    elementData = new int[100];
    size = 0;
}
```

  - This is ok, but not good style (LOSES POINTS)
    - We usually have one constructor that does most of the work
    - Other constructors call the main constructor (just like a method call)

```
public ArrayIntList() {
    this(100);
}
```

    - But this doesn't work:

```
public ArrayIntList() {
    this();
}
```

  - Finally, one last thing should make you uncomfortable
    - The 100
    - Why? It's arbitrary
    - We should make it a class constant
    - Why ok for class constant to be PUBLIC?
      - Because it cannot be changed