Warm Up: What is the output of this code?

```java
ArrayIntList list1 = new ArrayIntList();
ArrayIntList list2 = new ArrayIntList();
list1.add(1);
list2.add(2);
list1.add(3);
list2.add(4);
System.out.println(list1);
System.out.println(list2);
```
Recall: classes and objects

- **class**: A program entity that represents:
  - A complete program or module, or
  - A template for a type of objects.
  
  (ArrayList is a class that defines a type.)

- **object**: An entity that combines **state** and **behavior**.
  - **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects.
  - **abstraction**: Separation between concepts and details. Objects provide abstraction in programming.
Preconditions

- **precondition**: Something your method *assumes is true* at the start of its execution.
  - Often documented as a comment on the method's header:

    ```
    // Returns the element at the given index.
    // Precondition: 0 <= index < size
    public int get(int index) {
        return elementData[index];
    }
    ```

- Stating a precondition doesn't really "solve" the problem, but it at least documents our decision and warns the client what not to do.

- What if we want to actually enforce the precondition?
Throwing exceptions (4.4)

```java
throw new ExceptionType();
throw new ExceptionType("message");
```

- Generates an exception that will crash the program, unless it has code to handle ("catch") the exception.

- Common exception types:
  - ArithmeticException, ArrayIndexOutOfBoundsException, FileNotFoundException, IllegalArgumentException, IllegalStateException, IOException, NoSuchElementException, NullPointerException, RuntimeException, UnsupportedOperationException

- Why would anyone ever want a program to crash?
Postconditions

- **postcondition**: Something your method promises will be true at the end of its execution.
  - Often documented as a comment on the method's header:

    ```java
    // Precondition : size() < capacity
    // Postcondition: value is added at the end of the list
    public void add(int value) {
        elementData[size] = value;
        size++;
    }
    ```

- If your method states a postcondition, clients should be able to rely on that statement being true after they call the method.
**this keyword**

- **this**: A reference to the *implicit parameter* (the object on which a method/constructor is called)

**Syntax:**

- To refer to a field: `this.field`
- To call a method: `this.method(parameters);`
- To call a constructor from another constructor: `this(parameters);`
**ArrayList of primitives?**

- The type you specify when creating an `ArrayList` must be an object type; it cannot be a primitive type.

  ```java
  // illegal -- int cannot be a type parameter
  ArrayList<int> list = new ArrayList<int>();
  ```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

  ```java
  // creates a list of ints
  ArrayList<Integer> list = new ArrayList<Integer>();
  ```
Wrapper classes

- A wrapper is an object whose sole purpose is to hold a primitive value.

- Once you construct the list, use it with primitives as normal:

  ```java
  ArrayList<Double> grades = new ArrayList<Double>();
  grades.add(3.2);
  grades.add(2.7);
  ...
  double myGrade = grades.get(0);
  ```

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Tips for testing

- You cannot test every possible input, parameter value, etc.
  - Think of a limited set of tests likely to expose bugs.

- Think about boundary cases
  - Positive; zero; negative numbers
  - Right at the edge of an array or collection's size

- Think about empty cases and error cases
  - 0, -1, null; an empty list or array

- Test behavior in combination
  - Maybe add usually works, but fails after you call remove
  - Make multiple calls; maybe size fails the second time only