CSE 123

ArrayIntList

Questions during Class?
Raise hand or send here

sli.do  #cse123

BEFORE WE START

Talk to your neighbors:

Did you eat breakfast today? If so, what?

Music: 123 24su Lecture Tunes

Instructor: Joe Spaniac

TAs: Andras Daniel, Eric Nicole, Sahej Trien, Zach
Lecture Outline

- Announcements
- Arrays vs. ArrayLists
- ArrayIntList
  - Fields
  - Implementing add()
  - Capacity & Resizing
Announcements

• Check-in 1 “Graded”! (on gradescope)
• Quiz 1 Completed! 😊📚
  - Congrats! Expect grades back in about a week (hopefully)
  - Practice metacognition: how did that go? What can you learn about your studying process and how can you incorporate it before the next quiz?
• Programming Assignment 1 due tonight (7/3) @ 11:59pm
  - Try to get something in before the initial submission such that you can get feedback
  - Extra credit due (7/3) as well – totally ok if you don’t complete it!
• Creative Project 1 Grades out after lecture
• Resubmission period 1 closes on Friday (7/5) @ 11:59pm
  - Assignments available: C1
Lecture Outline

• Announcements

• **Arrays vs. ArrayLists**

• ArrayIntList
  - Fields
  - Implementing `add()`
  - Capacity & Resizing
## Arrays vs. ArrayLists

<table>
<thead>
<tr>
<th>Arrays</th>
<th>ArrayLists</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int[] arr = new int[x];</code></td>
<td><code>List&lt;Integer&gt; al = new ArrayList&lt;&gt;();</code></td>
</tr>
<tr>
<td><code>int y = arr[0]</code></td>
<td><code>int y = al.get(0);</code></td>
</tr>
<tr>
<td></td>
<td><code>al.add(2);</code></td>
</tr>
<tr>
<td><code>arr[0] = 5;</code></td>
<td><code>al.set(0, 5);</code></td>
</tr>
<tr>
<td><code>int length = arr.length;</code></td>
<td><code>int size = al.size();</code></td>
</tr>
<tr>
<td>// <strong>Always x</strong></td>
<td>// <strong>Matches # of things added</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fundamental data structure*</th>
<th>Class within java.util</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed length</td>
<td>Illusion of resizing</td>
</tr>
</tbody>
</table>

* Technically arrays are also Objects in Java, but for the purposes of this course / most of your CS career we’ll treat them like fundamental data structures
Lecture Outline

• Announcements

• Arrays vs. ArrayLists

• **ArrayIntList**
  - Fields
  - Implementing `add()`
  - Capacity & Resizing
Implementing Data Structures

• No different from designing any other class!
  - Specified behavior (List interface):

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(E value)</td>
<td>Adds the given value to the end of the list</td>
</tr>
<tr>
<td>add(int index, E value)</td>
<td>Adds the given value at the given index</td>
</tr>
<tr>
<td>remove(E value)</td>
<td>Removes the given value if it exists</td>
</tr>
<tr>
<td>remove(int index)</td>
<td>Removes the value at the given index</td>
</tr>
<tr>
<td>get(int index)</td>
<td>Returns the value at the given index</td>
</tr>
<tr>
<td>set(int index, int value)</td>
<td>Updates the value at the given index to the one given</td>
</tr>
<tr>
<td>size()</td>
<td>Returns the number of elements in the list</td>
</tr>
</tbody>
</table>

- Choose appropriate fields based on behavior

• Just requires some thinking outside the box
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size; // Storage boundary

al.add(2);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size; // Storage boundary

```java
al.add(2);
```
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;    // Where we store elements
  - int size;             // Storage boundary

al.add(2);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size;          // Storage boundary

al.add(5);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;  // Where we store elements
  - int size;          // Storage boundary

al.add(5);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;   // Where we store elements
  - int size;            // Storage boundary

al.add(5);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;    // Where we store elements
  - int size;            // Storage boundary

al.add(-1);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;     // Where we store elements
  - int size;              // Storage boundary

al.add(-1);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;    // Where we store elements
  - int size;             // Storage boundary

al.add(-1);
Lecture Outline

• Announcements

• Arrays vs. ArrayLists

• ArrayIntList
  - Fields
  - Implementing add()
  - Capacity & Resizing
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;  // Where we store elements
  - int size;           // Storage boundary

al.add(0, 0);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size; // Storage boundary

al.add(0, 0);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size; // Storage boundary

al.add(0, 0);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - `int[] elementData;` // Where we store elements
  - `int size;` // Storage boundary

```java
al.add(0, 0);
```
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;  // Where we store elements
  - int size;  // Storage boundary

```java
al.add(0, 0);
```
ArrayIntLists

- For simplicity: only about storing ints (no type variables)
- How do we accomplish resizing magic trick? Two fields:
  - int[] elementData; // Where we store elements
  - int size; // Storage boundary

al.add(0, 0);
ArrayIntLists

• For simplicity: only about storing ints (no type variables)
• How do we accomplish resizing magic trick? Two fields:
  - int[] elementData;    // Where we store elements
  - int size;             // Storage boundary

• Important points:
  - size represents how far the curtain is peeled back
    - Can’t use a hardcoded value!
  - Starting value is always at index 0
    - Adding to / removing from beginning requires shifting elements
Lecture Outline

• Announcements

• Arrays vs. ArrayLists

• ArrayIntList
  - Fields
  - Implementing add()
  - Capacity & Resizing
Capacity and Resizing

• Capacity = length of underlying array
• Size = number of user-added elements
• What happens if we run out of space? (size == capacity)

```
al.add(2);
```
### Capacity and Resizing

- **Capacity** = length of underlying array
- **Size** = number of user-added elements
- What happens if we run out of space? ($\text{size} == \text{capacity}$)
Capacity and Resizing

- **Capacity** = length of underlying array
- **Size** = number of user-added elements
- What happens if we run out of space? (size == capacity)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>5</td>
<td>12</td>
<td>-9</td>
<td>7</td>
<td>21</td>
<td>-3</td>
</tr>
</tbody>
</table>

B)
Capacity and Resizing

• Capacity = length of underlying array
• Size = number of user-added elements
• What happens if we run out of space? (size == capacity)
Capacity and Resizing

• Capacity = length of underlying array
• Size = number of user-added elements
• What happens if we run out of space? (size == capacity)

B)

```
0  2  5  -1  5  12  -9  7  21  -3
```

`al.add(2);`
Capacity and Resizing

- Capacity = length of underlying array
- Size = number of user-added elements
- What happens if we run out of space? (size == capacity)

```
B)
0  1  2  3  4  5  6  7  8  9  10
0  2  5  -1  5  12 -9  7  21 -3  0
```

```java
al.add(2);
```
Capacity and Resizing

• Capacity = length of underlying array
• Size = number of user-added elements
• What happens if we run out of space? (size == capacity)

```
B)

0 1 2 3 4 5 6 7 8 9 10
0 2 5 -1 5 12 -9 7 21 -3 2
al.add(2);
```
Capacity and Resizing

• Capacity = length of underlying array
• Size = number of user-added elements
• What happens if we run out of space? (size == capacity)
  - We make a new (bigger array) and copy things over
  - Another layer to the resizing illusion!

• In reality, we don’t typically add a single spot
  - What happens if we add again?