Announcements/Follow-ups

• Final Exam 2 weeks from today
• Follow-ups
  – Questions (midterm 2, lab 8, quiz 4)
  – For each loop mistake
    • Updated slides and examples will be posted
  – Finish generics
For-each loop mistake

This doesn’t modify the elements!

ArrayList<Double> myDoubleCollection;
...
for(Double x : myDoubleCollection) {
    x += 100.0;
}
• ForEachAndIterators example revisited
For-each loop mistake

This would modify the elements:

```java
ArrayList<MutableDouble> myMutables;
...
for(MutableDouble x : myMutables) {
    x.increaseValueBy(100.0);
}
```
ArrayList<Integer> myList;
for(Integer myInt : myList)
myInt++; // myInt = myInt + 1;
for (Integer myInt : myList)
myInt++; // myInt = myInt + 1;
for(Integer myInt : myList)
myInt++; // myInt = myInt+1;

Stack

myList @

myInt @

Heap

ArrayList object

@ @ @

Integer object 1

Integer object 5

Integer object 9

Integer object 10
Generics

• Wednesday Slides
  – Implementing generic methods
  – Implementing generic interfaces
  – Collections algorithms

• Generics don’t play well with arrays
  – GenericSwap revisited
  – When using generics, use collections

• LexArray example
Data Structures

• Any way of organizing data in memory
  – Objects
  – Arrays
  – Linked lists
  – Collections
  – Databases

• Typically optimized for a specific purpose
  – Inserting new data
  – Finding existing data (“queries”)
  – Etc.
Linear search

10?
Linear search on sorted array

10?
Binary search on sorted array

10?

<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>
</table>
Binary search on sorted array

10?
Binary search on sorted array

10?
Binary search on sorted array

10?

• BinarySearch example
Binary search tree

Root: 9

3

1 7

4 8

16

11 22

10 > 9

10 < 16

Leaf!
Binary tree

• Each entry is called a “node” or “vertex”
• Arrows connecting nodes are called “edges”
• Arrows point from “parent” nodes to “child” nodes
• Each node has two “children”, left and right.
• Children of children are called “descendants”
• All descendents of a node are called a “sub-tree”
• The top node is called the “root”
• Nodes without children are called “leaves”
Binary search tree

- Every node in the left sub-tree has a value less than the root
- Every node in the right sub-tree has a value greater than the root
- To “search” the tree:
  - Start at the root
  - Compare the query with the root value
  - Move left or right depending on the comparison
  - Repeat on the resulting sub-tree
  - Stop when the value is found, or a leaf is reached
Binary search tree – with array

```
0  1  2   3  4   5  6
9  3  16  1   7  11  12
22
```
Binary search tree – with array

```
   9
  /   \
 3     16
 /   /   \
1   7   11
   /   \
  22
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>16</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Binary search tree – with array

```
0  9  1  3  2  16  3  1  4  7  5  11  6  22
```
Binary search tree – with array

\[ \ell(i) \quad ? \quad r(i) \]
Binary search tree – with array

<table>
<thead>
<tr>
<th>Parent</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

0

1 2

3 4 5 6

7 8 9 10 11 12 13 14
Binary search tree - references

Tree object
- data: 1
- left: null
- right: null

Tree object
- data: 0
- left: null
- right: null

Tree object
- data: 3
- left: null
- right: null
Hash Tables

• A mapping from (potentially non-integer) keys to values.
  – E.g. White pages
• Supports fast value look-up for a given key
• Keys are mapped to a set of buckets
  – Typically they should be distributed evenly
• Look-up by jumping directly to the correct bucket
• MyHash example
Hash Tables

\[
\begin{align*}
\text{k \% 3 == 0} & \quad (0, "Hi") \quad (3, "There") \quad (99, "How") \\
\text{k \% 3 == 1} & \quad (13, "Are") \\
\text{k \% 3 == 2} & \quad (5, "You") \quad (101, "Bog")
\end{align*}
\]
Hash Tables

(0, “Hi”)  (3, “There”)  (99, “How”)
(13, “Are”)
(5, “You”)  (101, “Bog”)

Lookup key 101

101 % 3 == 2