Operations:
//returns true if x in tree, o/w returns false

boolean contains (Object x, BinaryNode t)

Idea:
contains(3)?

Start at root

For each node, check element.

If = x , done! return true

If > x, recurse on left child

If < x, "right"

Base Case? When node is null, return false

contains (Object x, BinaryNode t)?

if t == null
    return false
else:
    if t.element == x
        return true
    else if t.element > x
        return contains (x, t.left)
    else // t.element < x
        return contains (x, t.right)
}
\textbf{ex0} contains (4, root) = contains (4, 6) \quad 6 > 4 \Rightarrow \text{left}

\quad \uparrow

\quad \text{contains (4, 2)} \quad 2 < 4 \Rightarrow \text{right}

\quad \uparrow

\quad \text{contains (4, 4)} \quad 4 = 4 \Rightarrow \text{return true}

\textbf{ex2} contains (5, 6) \quad 6 > 5 \Rightarrow \text{left}

\quad \downarrow

\quad \text{contains (5, 2)} \quad 2 < 5 \Rightarrow \text{right}

\quad \downarrow

\quad \text{contains (5, 4)} \quad 4 < 5 \Rightarrow \text{right}

\quad \downarrow

\quad \text{contains (5, null)} \quad t = \text{null}, \text{return false}

\underline{Run Time?} \quad \text{For a tree with } n \text{ nodes}

\underline{Worst-case}: \text{start at root, keep searching until we hit a leaf \textit{(traverse the depth of tree)}}

\underline{Worst-case depth?} \quad \text{Average Case is } \log(n).

\begin{itemize}
  \item Top level has 1 node (root)
  \item Each level has \(2^n\) twice as many nodes as previous level
  \item Bottom level has \(\frac{n}{2} \text{ nd leafs}
\end{itemize}
\begin{align*}
\text{depth} &= \# \text{ levels} \\
&= \# \text{ times we can double} \\
&\quad \text{from } 1 \text{ until } \frac{n}{2} \\
&= O(\log \left( \frac{n}{2} \right)) = O(\log(n)) \\
\end{align*}

```
\text{Runtime} = O(\text{depth}) = \text{worst-case: } O(n) \\
\quad \text{avg-case: } O(\log(n))
```

Similar for most operations

Next operation: insert

Similar to contains.

```
\begin{array}{c}
6 \\
4 \\
2 \\
1 \\
3 \\
5 \\
\end{array}
```

ex: insert(5)

```
\begin{array}{c}
6 \\
4 \\
2 \\
1 \\
3 \\
5 \\
\end{array}
```

Where should it go? 

only one spot!

```
\begin{array}{c}
6 \\
4 \\
2 \\
1 \\
3 \\
5 \\
\end{array}
```

Idea: Search for element (contains()), if not in tree, insert.

Where? Child of last (non-null) node visited

will have another version of contains() that returns last non-null node visited.

(modified-contains)
// One implementation approach

insert(Object e, BinaryNode t) { // insert element e in tree with root t

    // create new node
    BinaryNode n = new BinaryNode(e)

    // Get last visited non-null node in search for e
    BinaryNode p = modifiedContains(e, t);

    // pseudo-code for compareTo()
    if (p.element < e) // add to left of p
        p.left = n
    else if (p.element > e) // add to right of p
        p.right = n
    else { // p.element == e (duplicate!)
        p.addDuplicate(e)
    }

    p = 4  \Rightarrow  4 < 5, right

What about duplicates?

every node has duplicate array
(add to BinaryNode class?)

Note: Some implementations don't do this!

\textbf{Fig. 5: insert (5, 6)}
Keep in mind elements may not just be numbers:
We can store any objects.

Now that we're considering a data structure
that orders the elements, it's unclear how to order
objects.

ex: BST of BankAccounts, Persons, Airlines
(Each of these objects have many variables)

Make an object "comparable."

1. Assign one data variable as key - used for comparison
   (your choice).
2. Write a compareTo method using key (Java-recognized
   method like toString())
3. Make class "implement Comparable"

1. ex. BankAccount key: balance

2. public int compareTo (BankAccount other)

   if (balance == other.balance)
       return 0
   else if (balance < other.balance)
       return -1
   else // balance > other balance
       return 1
public class BankAccount implements Comparable<BankAccount>

Sample Code: BankAccount, Person
HW #5 Pseudocode

For each file in the directory
   For each word in the file:
       Create word object, w
       - string the word
       - filename into file list
   If w not in tree
       insert it
   else
       retrieve it from tree
       update file list to include filename.