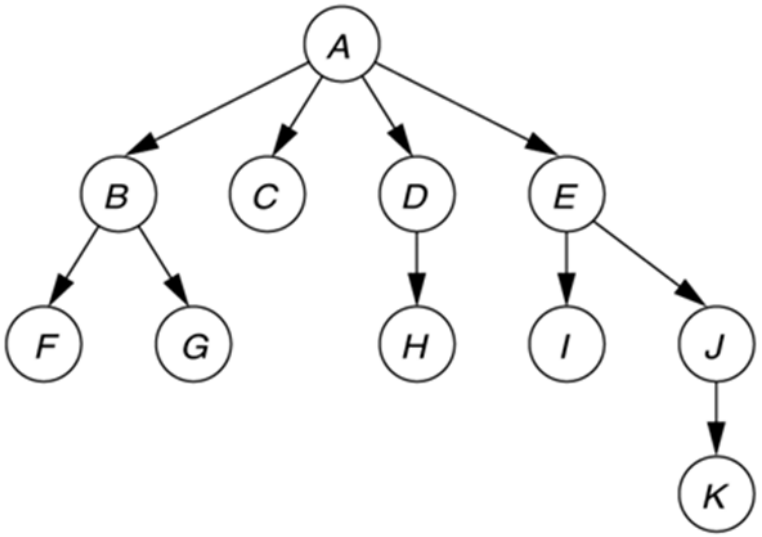
Q1

CSSE 230 Day 7

More simple BinaryTree methods

Tree Traversals



Questions?

Dr. B's quiz: What became clear to you as a result of class?

CSSE230: student I was **tree**ted to some good knowledge by the time I **leaf**t.

Agenda

- Implementing Binary Trees
- Binary Tree Traversals
- Binary Tree Iterators

Growing Trees

Let's continue implementing a *BinaryTree<T>* class including methods *size()*, *height()*, *duplicate()*, and *contains(T)*.

Binary tree traversals

- PreOrder (top-down, depth-first)
  - root, left, right
- PostOrder (bottom-up)
  - left, right, root
- InOrder (left-to-right, if tree is spread out)
  - Left, root, right
- LevelOrder (breadth-first)
  - Level-by-level, left-to-right within each level

If the tree has N nodes, what's the (worst-case) big-Oh run-time of each traversal? big-Oh space used?

```
// Print tree rooted at current node using preorder
public void printPreOrder( )  {
    System.out.println( element );        // Node
    if( left != null )
        left.printPreOrder( );            // Left
    if( right != null )
        right.printPreOrder( );           // Right
}


// Print tree rooted at current node using postorder
public void printPostOrder( ) {
    if( left != null )
        left.printPostOrder( );           // Left
    if( right != null )
        right.printPostOrder( );          // Right
    System.out.println( element );        // Node
}

// Print tree rooted at current node using inorder t
public void printInOrder( ) {
    if( left != null )
        left.printInOrder( );             // Left
    System.out.println( element );        // Node
    if( right != null )
        right.printInOrder( );            // Right
}
```

Java Collections Framework

Available, efficient, bug-free implementations of many key data structures

Most classes are in java.util

Weiss Chapter 6 has more details about collections

What's an iterator?

- In Java, specified by *java.util.Iterator<E>*

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

- *ListIterator<E>* adds:

boolean	**hasPrevious**() Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
int	**nextIndex**() Returns the index of the element that would be returned by a subsequent call to `next`.
Object	**previous**() Returns the previous element in the list.
int	**previousIndex**() Returns the index of the element that would be returned by a subsequent call to `previous`.
void	**set**(Object o) Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

Using an Iterator

ag can be any `Collection` of `Integers`

```
for (Integer val : ag)
    sum += val;
System.out.println(sum);
```

The Java compiler translates the above into this:

```
for (Iterator<Integer> itr = ag.iterator(); itr.hasNext(); )
 sum += itr.next();
System.out.println(sum);
```

Binary Tree Iterators

What if we want to iterate over the elements in the nodes of the tree one-at-a-time instead of just printing all of them?