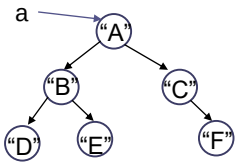
IMPLEMENTING BINARY TREES

Chapter 6

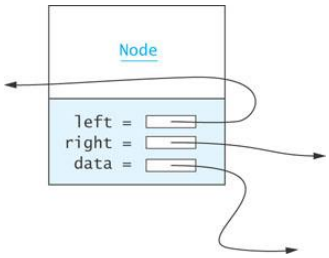
A Binary Tree

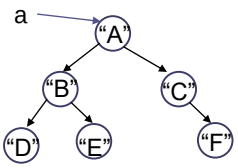
BinaryTree<String> a = new BinaryTree<String>();



A Binary Tree

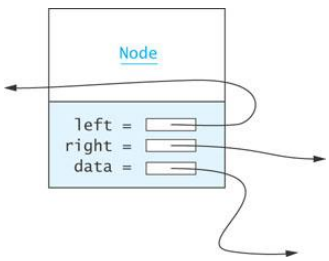
BinaryTree<String> a = new BinaryTree<String>();

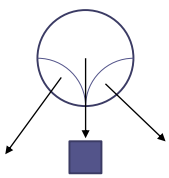


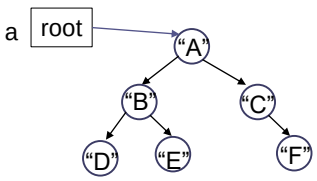


A Binary Tree

BinaryTree<String> a = new BinaryTree<String>();



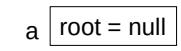


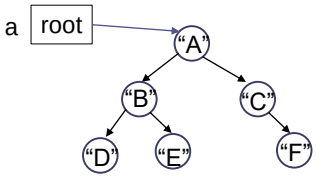


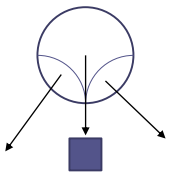
A Binary Tree

BinaryTree<String> a = new BinaryTree<String>();

An Empty Tree







Implementing a `BinaryTree` Class

Section 6.3

BinaryTree<E> Class (cont.)

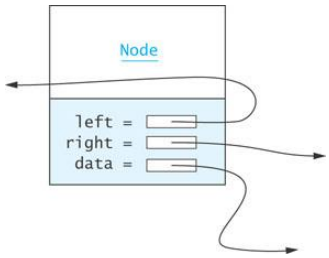
```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

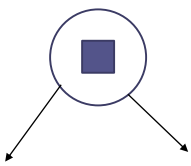
BinaryTree<E> Class (cont.)

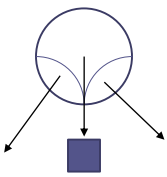
```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

Node<E> Class

- Just as for a linked list, a node consists of a data part and links to successor nodes
- The data part is a reference to type E
- A binary tree node must have links to both its left and right subtrees

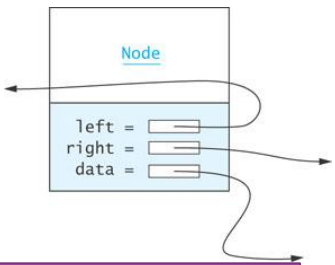






Node<E> Class (cont.)

```
public class Node<E> {
   public E data;
   public Node<E> left;
   public Node<E> right;

   public Node(E data) {
     this.data = data;
     left = null;
     right = null;
   } // Node()

   public String toString() {
      return data.toString();
   } // toString()
} class Node<E>
```



Node<E> is declared as an inner class within BinaryTree<E>

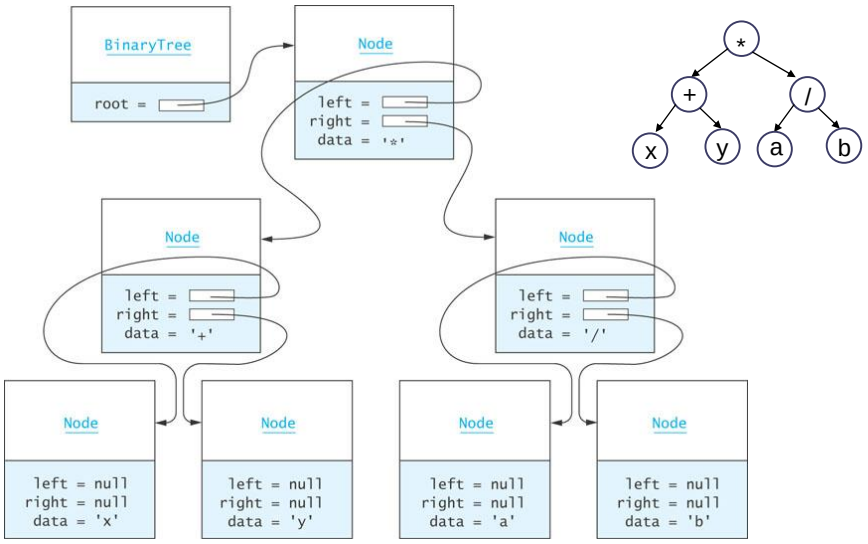
BinaryTree<E> Class (cont.)

```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

BinaryTree<E> Class (cont.)

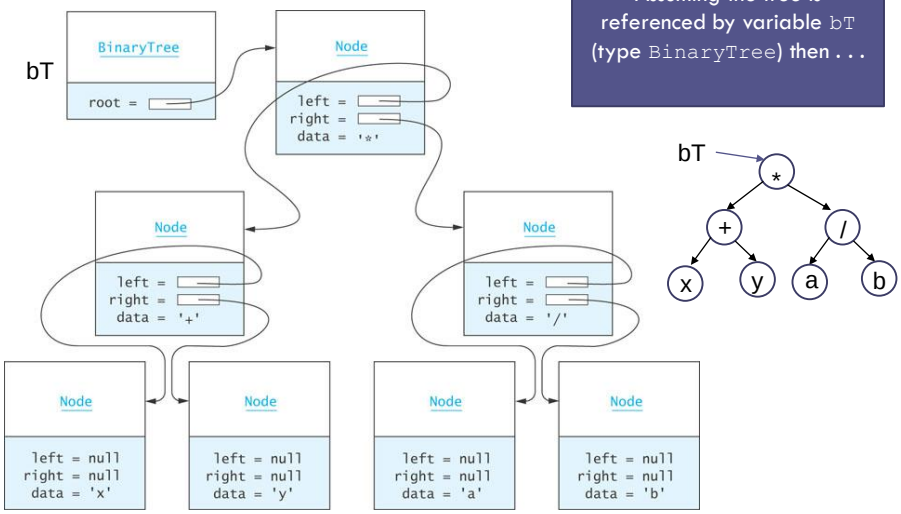
```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

BinaryTree<E> Class (cont.)

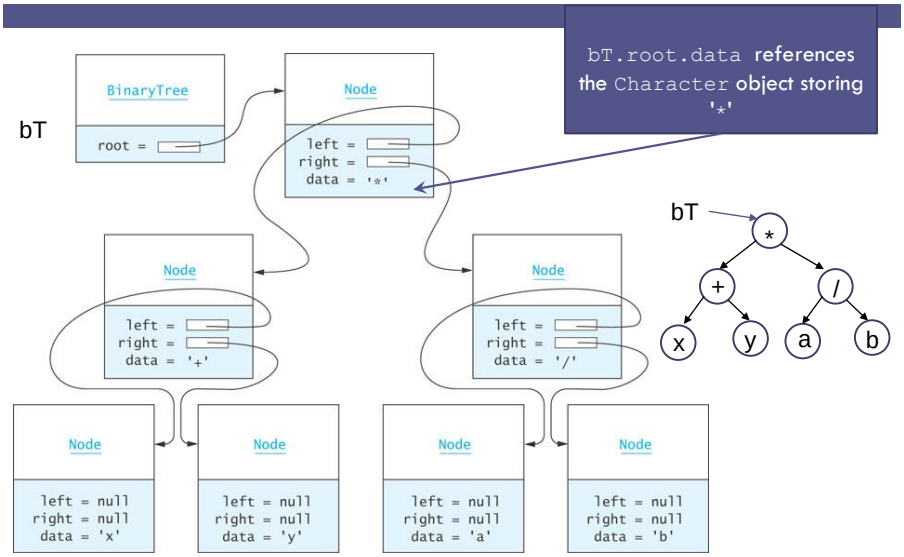


BinaryTree<E> Class (cont.)

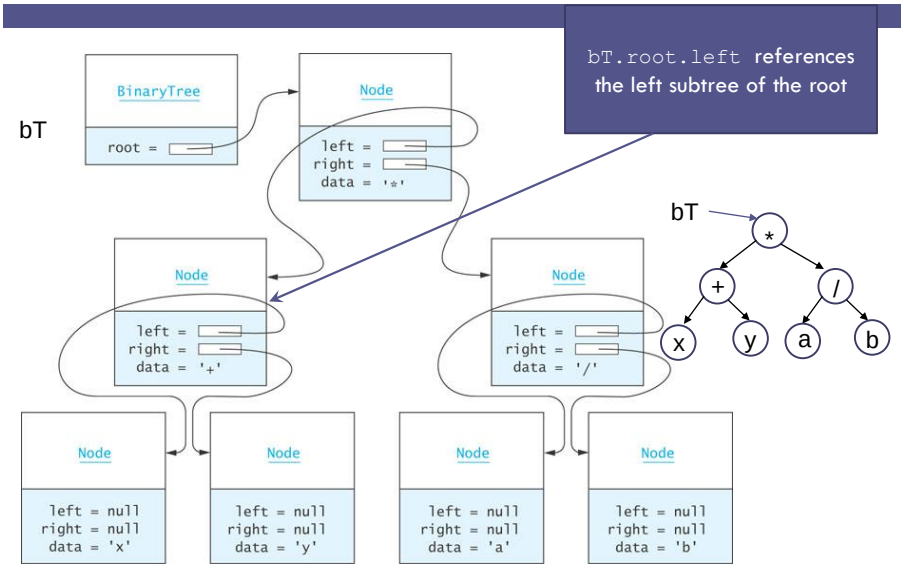
Assuming the tree is referenced by variable `bT` (type `BinaryTree`) then . . .



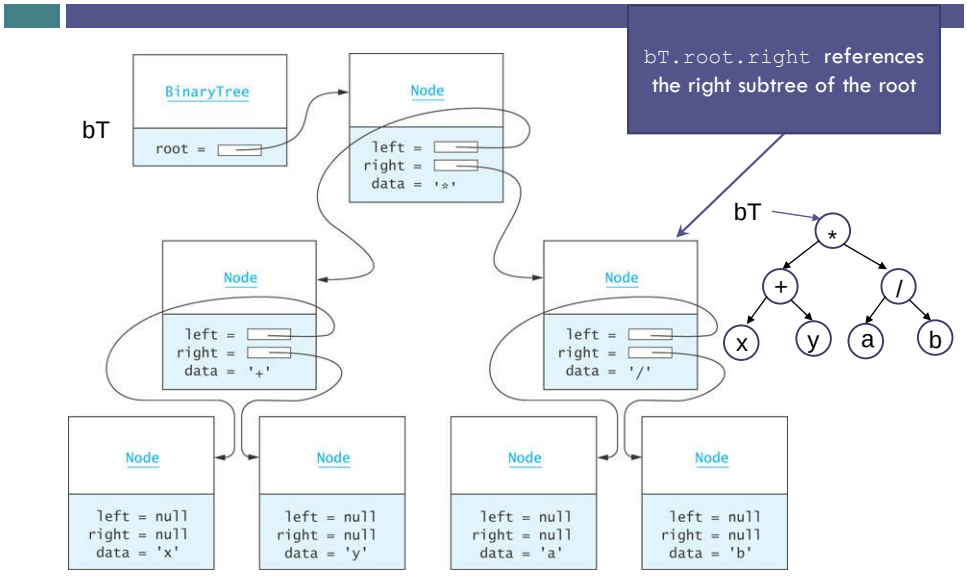
BinaryTree<E> Class (cont.)



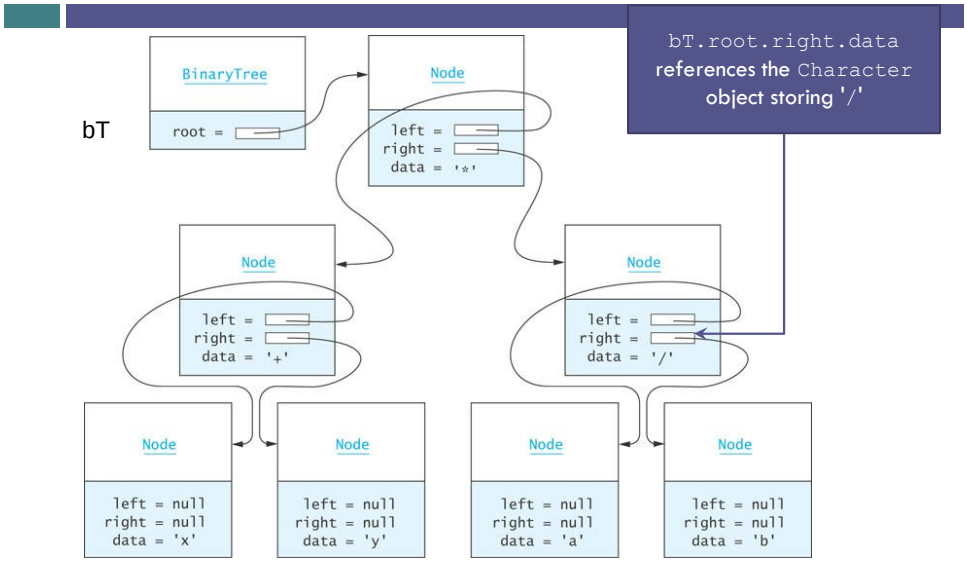
BinaryTree<E> Class (cont.)



BinaryTree<E> Class (cont.)



BinaryTree<E> Class (cont.)



BinartTree<E> Class

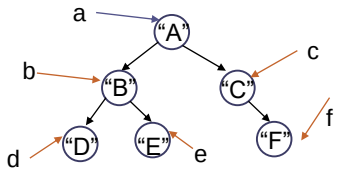
```
// Constructors
public BinaryTree() {
   root = null;
} // BinaryTree()

public BinaryTree(Node<E> root) {
   this.root = root;
} // BinaryTree()
```

BinartTree<E> Class

```
// Constructors
public BinaryTree(E data, BinaryTree<E> leftTree,
                          BinaryTree<E> rightTree) {

   root = new Node<E>(data);
   if (leftTree != null) {
      root.left = leftTree.root;
   } else {
      root.left = null;
   }
   if (rightTree != null) {
      root.right = rightTree.root;
   } else {
      root.right = null;
   }
} // BinaryTree()
```

Using BinaryTree<E> Class

```
public static void main(String[] args) {
   // Create some Binary trees to put together
   // a tree [a [b [d][e]], [c [][f]]]

   BinaryTree<String> d = new BinaryTree<String>("D", null, null);
   BinaryTree<String> e = new BinaryTree<String>("E", null, null);
   BinaryTree<String> f = new BinaryTree<String>("F", null, null);

   BinaryTree<String> b = new BinaryTree<String>("B", d, e);
   BinaryTree<String> c = new BinaryTree<String>("C", null, f);
   BinaryTree<String> a = new BinaryTree<String>("A", b, c);

   System.out.println(a);
} // main()
```



BinaryTree<E> Class (cont.)

```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;

   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)

   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

BinaryTree<E> Class (cont.)

```
public class BinaryTree<E> {
  // Data members/fields...just the root is needed
  - Node<E> root;
  // Constructor(s)
  + BinaryTree()
  + BinaryTree(Node<E> node)
  + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
  // Methods: Accessors
  + E getData() throws BinaryTreeException
  + boolean isEmpty()
  + void clear()
  + BinaryTree<E> getLeftSubtree()
  + BinaryTree<E> getRightSubtree()
  + boolean isLeaf()
  + int height()
  - int height(Node<E> node)
  + String toString()
  - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {
} // BinaryTree<E> class
```

BinartTree<E> Class

```
// Constructors
public boolean isEmpty() {
    return this.root==null;
} // isEmpty()

public void clear() {
    this.root = null;
} // clear()
```

BinaryTree<E> Class (cont.)

public class BinaryTree<E> {

// **Data members/fields…just the root is needed**
- Node<E> root;

// **Constructor(s)**
+ BinaryTree()
+ BinaryTree(Node<E> node)
+ BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)

// **Methods: Accessors**
+ E getData() throws BinaryTreeException
+ boolean isEmpty()
+ void clear()
+ BinaryTree<E> getLeftSubtree()
+ BinaryTree<E> getRightSubtree()
+ boolean isLeaf()
+ int height()
- int height(Node<E> node)
+ String toString()
- void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {

} // BinaryTree<E> class

BinaryTree<E> Class (cont.)

```
public class BinaryTreeException extends Exception {

    BinaryTreeException(String message) {
      super(message);
    } // BinaryTreeException()

} // class BinaryTreeException

public E getData() {
   if (this.root == null)
      throw new BinaryTreeException("Accessing data in null tree");

   return this.root.data;
} // getData()
```

BinaryTree<E> Class (cont.)

public class BinaryTree<E> {

// **Data members/fields...just the root is needed**
- Node<E> root;

// **Constructor(s)**
+ BinaryTree()
+ BinaryTree(Node<E> node)
+ BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)

// **Methods: Accessors**
+ E getData() throws BinaryTreeException
+ boolean isEmpty()
+ void clear()
+ BinaryTree<E> getLeftSubtree()
+ BinaryTree<E> getRightSubtree()
+ boolean isLeaf()
+ int height()
- int height(Node<E> node)
+ String toString()
- void preOrderTraverse(Node<E> node, int depth, StringBuilder sb) {

} // BinaryTree<E> class

BinaryTree<E> Class

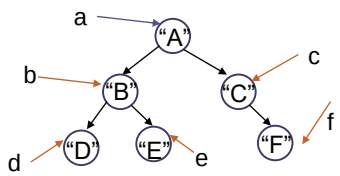
```
public BinaryTree<E> getLeftSubTree() {
   if (root != null && root.left != null)
      return new BinaryTree<E>(root.left);
   else
      return new BinaryTree<E>();
} // getLeftSubTree()

public boolean isLeaf() {
   return (root == null) ||
   (root.left==null && root.right==null);
} // isLeaf()
```

BinaryTree<E> Class (cont.)

```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb)
} // BinaryTree<E> class
```

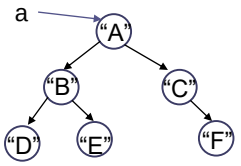
Using BinaryTree<E> Class

```
public static void main(String[] args) {
   // Create some Binary trees to put together
   // a tree [a [b [d][e]], [c [][f]]]

   BinaryTree<String> d = new BinaryTree<String>("D", null, null);
   BinaryTree<String> e = new BinaryTree<String>("E", null, null);
   BinaryTree<String> f = new BinaryTree<String>("F", null, null);
   BinaryTree<String> b = new BinaryTree<String>("B", d, e);
   BinaryTree<String> c = new BinaryTree<String>("C", null, f);
   BinaryTree<String> a = new BinaryTree<String>("A", b, c);

   System.out.println(a);
} // main()
```



What to print?

```
A
  B
    D
      null
      null
    E
      null
      null
  C
    null
    F
      null
      null
```



BinaryTree<E> Class

```
public String toString() {
   StringBuilder sb = new StringBuilder();
   preOrderTraverse(this.root, 1, sb);
   return sb;
} // toString()
```

BinaryTree<E> Class

```
public String toString() {
   StringBuilder sb = new StringBuilder();
   preOrderTraverse(this.root, 1, sb);
   return sb;
} // toString()


private void preOrderTraverse(Node<E> node, int depth,
                                                StringBuilder sb) {
   for (int i=0; i < depth; i++)
      sb.append("  ");

   if (node == null)
      sb.append("null\n");
   else {
      sb.append(node.toString());
      sb.append("\n");
      preOrderTraverse(node.left, depth+1, sb);
      preOrderTraverse(node.right, depth+1, sb);
   }
} // preOrderTraverse()
```

BinaryTree<E> **Class** (cont.)

public class BinaryTree<E> {

// **Data members/fields...just the root is needed**
- Node<E> root;

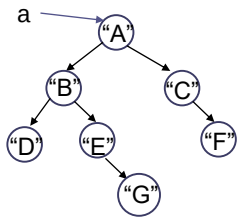
// **Constructor(s)**
+ BinaryTree()
+ BinaryTree(Node<E> node)
+ BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)

// **Methods: Accessors**
+ E getData() throws BinaryTreeException
+ boolean isEmpty()
+ void clear()
+ BinaryTree<E> getLeftSubtree()
+ BinaryTree<E> getRightSubtree()
+ boolean isLeaf()
+ int height()
- int height(Node<E> node)
+ String toString()
- void preOrderTraverse(Node<E> node, int depth, StringBuilder sb)

} // BinaryTree<E> class

Height of a Binary Tree

$$height(tree) = \begin{cases} 0 & if\ tree\ is\ empty \\ 1 + \max(height(leftSubtree), height(rightSubtree) \end{cases}$$



Height of a Binary Tree

$$height(tree) = \begin{cases} 0\ if\ tree\ is\ empty \\ 1 + \max(height(leftSubtree), height(rightSubtree) \end{cases}$$

```
public int height() {
   return height(root);
} // height()

private int height(Node<e> node) {
   if (node == null)
      return 0;
   else
      return 1 + Math.max(height(node.left), height(node.right));
} // height()
```

BinaryTree<E> **Class** (cont.)

```
public class BinaryTree<E> {
   // Data members/fields...just the root is needed
   - Node<E> root;
   // Constructor(s)
   + BinaryTree()
   + BinaryTree(Node<E> node)
   + BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree)
   // Methods: Accessors
   + E getData() throws BinaryTreeException
   + boolean isEmpty()
   + void clear()
   + BinaryTree<E> getLeftSubtree()
   + BinaryTree<E> getRightSubtree()
   + boolean isLeaf()
   + int height()
   - int height(Node<E> node)
   + String toString()
   - void preOrderTraverse(Node<E> node, int depth, StringBuilder sb)
} // BinaryTree<E> class
```