Tree Implementation Strategy

- Benefit of no node class
  - Simplicity
  - Implement all operations recursively
- Benefit node class
  - Conceptually can think of nodes
  - Option to encapsulate functionality within the node(getHeight, numberOfNodes, etc.)
- Benefit of separate node class
  - Reusable code at the cost of more complicated design

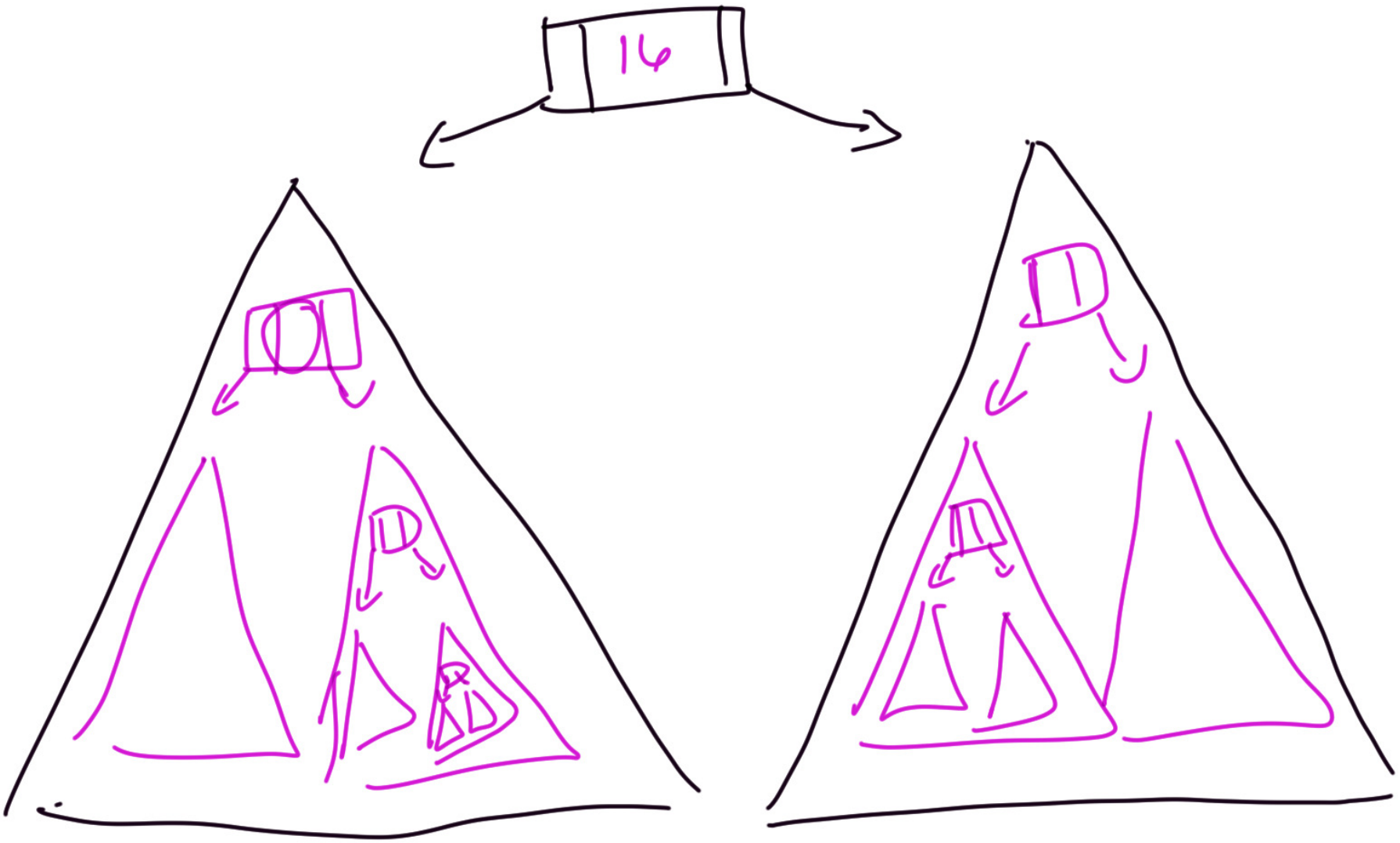
Without a distinct node class

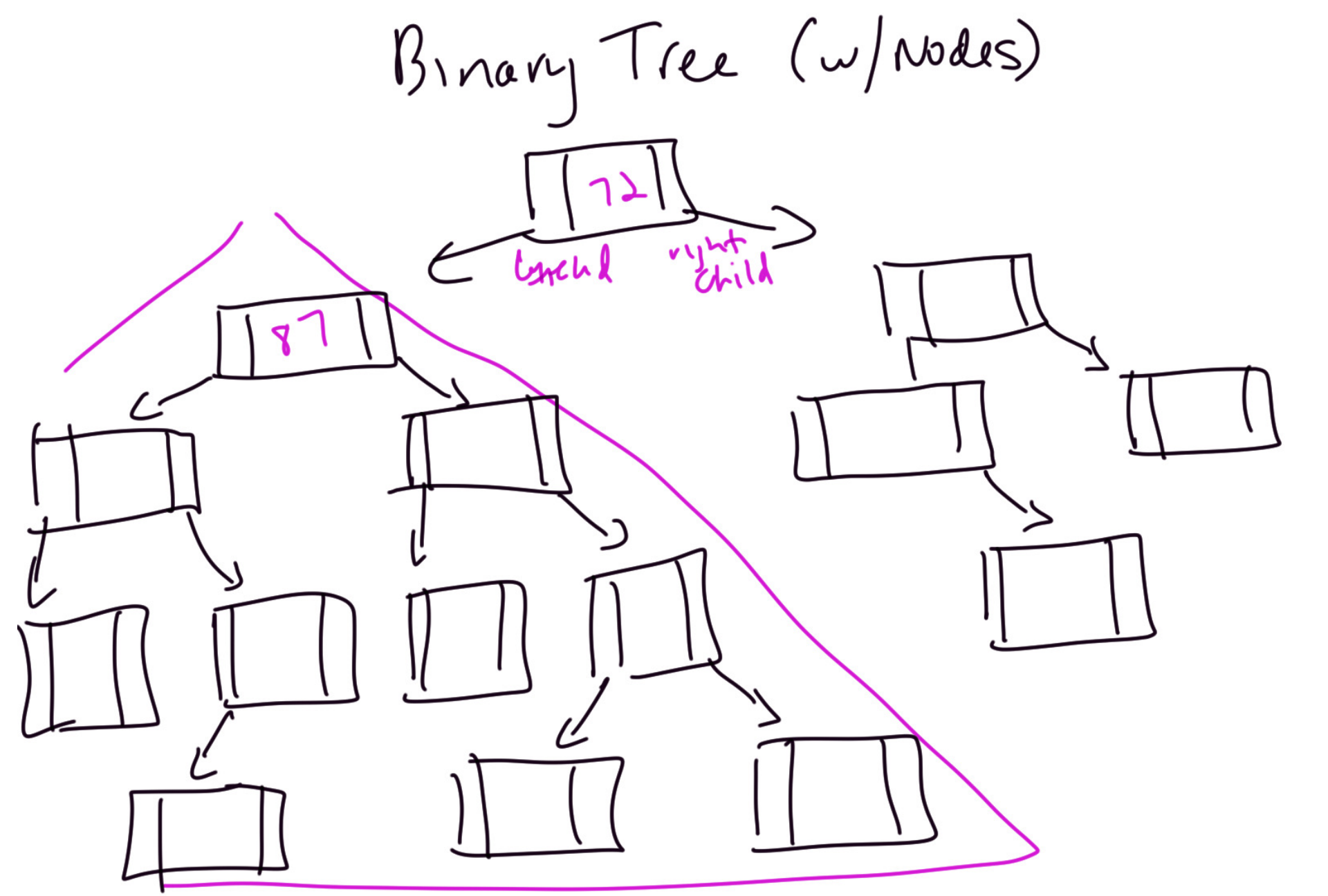
```
public class BinaryTree<T>
{
   //~ Instance/static variables .....................................
   private T element;
   private BinaryTree<T> left;
   private BinaryTree<T> right;

   //~ Constructors ..................................................
   /** Creates a single binary tree node containing the
      * given element and no children.
      *  @param value The data element to store in the new tree node.
    */
   public BinaryTree(T value)
   {
      element = value;
      left = null;
      right = null;
   }

  /* * Creates a single binary tree node containing the given
    * element and child subtrees.
    * @param value The data value to store on the new node.
    * @param leftChild  A reference to the left child for the new node.
    * @param rightChild A reference to the right child for the new node.
    */
   public BinaryTree(
      T value, BinaryTree<T> leftChild, BinaryTree<T> rightChild)
   {
      element = value;
      left = leftChild;
      right = rightChild;
}
```

Binary Tree (of trees)




Binary Tree (w/Nodes)
72
left child
right child
87

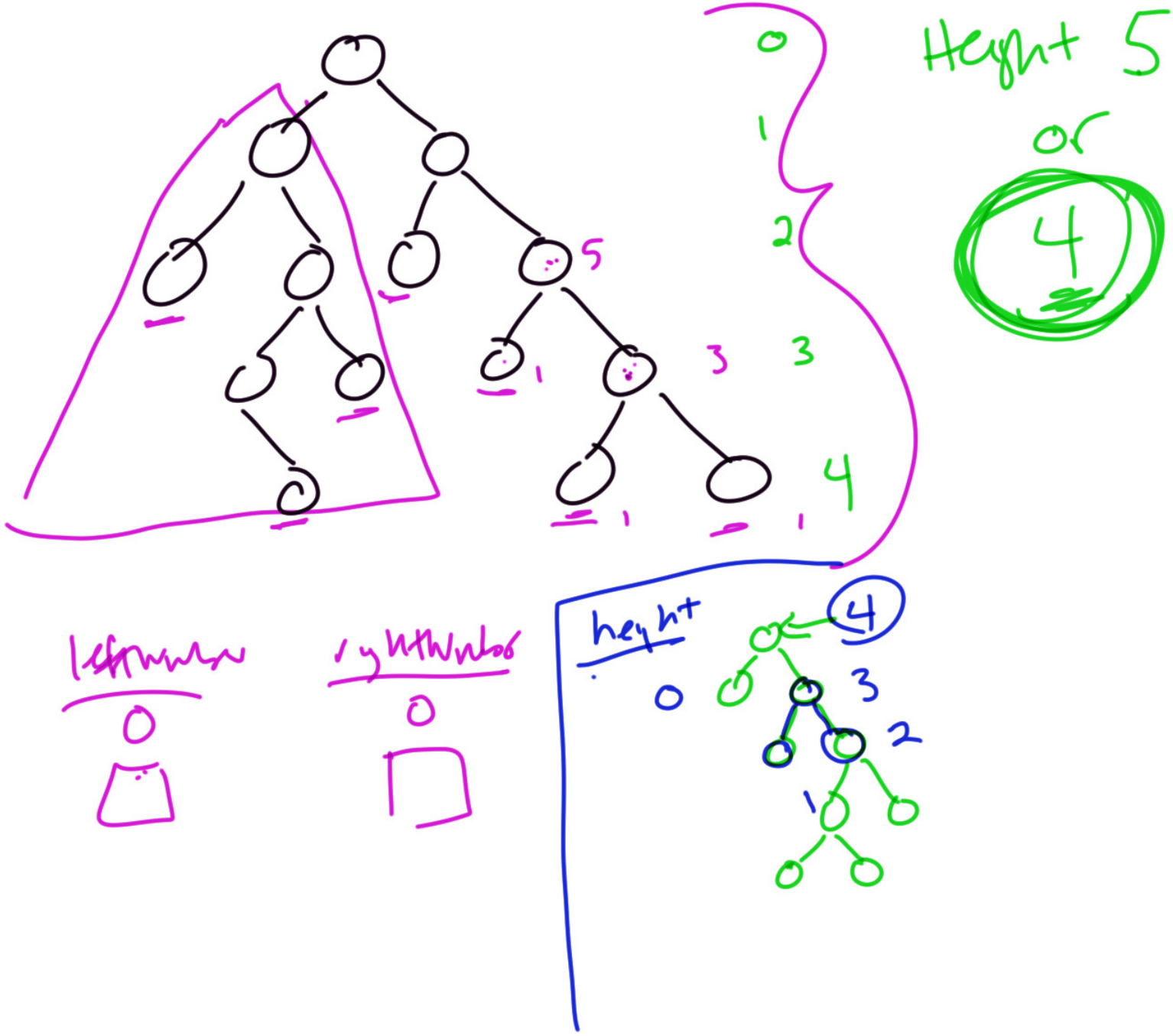
```
11 class BinaryNode<T> {
12
13     private T data;
14     private BinaryNode<T> leftChild;
15     private BinaryNode<T> rightChild;
16
17     /**
18      * Creates a node with no children.
19      *
20      * @param theElement the element to store in this node.
21      */
22     BinaryNode(T entry) {
23         data = entry;
24         leftChild = null;
25         rightChild = null;
26     }
27
```

```
4  public class BinaryTree<T> {
5      private BinaryNode<T> root;
6
7      /**
8       * Constructs an empty tree.
9       */
10     public BinaryTree() {
11         root = null;
12     }
13
14     /**
15      * Constructs an empty tree.
16      */
17     public BinaryTree(BinaryNode<T> topNode) {
18         root = topNode;
19     }
20
```

```
37      /**
38       * Counts the nodes in the tree
39       *
40       * @return The number of nodes
41       */
42      public int getNumberOfNodes() {
43          return getNumberOfNodes(root);
44      }
45
46      /**
47       * Returns the height of the tree
48       *
49       * @return The number of nodes
50       */
51      public int getHeight() {
52          return getHeight(root);
53      }
54
```

```
55    private int getNumberOfNodes(BinaryNode<T> node) {
56        int leftNumber = 0;
57        int rightNumber = 0;
58
59        if (node.getLeft() != null)
60            leftNumber = getNumberOfNodes(node.getLeft());
61
62        if (node.getRight() != null)
63            rightNumber = getNumberOfNodes(node.getRight());
64
65        return 1 + leftNumber + rightNumber;
66    } // end getNumberOfNodes
67
```

```
68      private int getHeight(BinaryNode<T> node) {
69          int height = 0;
70
71          if ((node != null) && (!node.isLeaf()))
72              height = 1 + Math.max(getHeight(node.getLeft()),
73                                    getHeight(node.getRight()));
74
75          return height;
76      } // end getHeight
77
```


Height 5
or
4
0
1
2
3
3
4
5
1
1
1
height
0
4
3
2
1