<https://github.com/mdabarik/binary-tree>

Difference between Tree and Binary Tree

**🔹 Main Differences**

| **Feature** | **General Tree** | **Binary Tree** |
| --- | --- | --- |
| Children per node | Any number (0..N) | At most 2 (0, 1, or 2) |
| Node distinction | No left/right concept | Left child & Right child |
| Complexity | More flexible, less structured | More structured, useful in many algorithms |
| Examples / Uses | File systems, XML/HTML DOM | BST, Heaps, Expression Trees |

What is Binary Tree?

If each node has zero, one or max two nodes.

|  |  |  |
| --- | --- | --- |
| Left Node | Data | Right Node |

Public class TreeNode<T> {

T data;

TreeNode left;

TreeNode right;

TreeNode(T data){

This.data=data;

}

PreOrder Binary tree traversal

Root->left->right

Diffrence between Binary Tree and Red black tree?

**Key Differences: BST vs Red-Black Tree**

| **Feature** | **BST** | **Red-Black Tree (RBT)** |
| --- | --- | --- |
| Structure | Binary tree | Binary tree + color info |
| Balancing | Not guaranteed | Self-balancing rules |
| Worst-case height | n (skewed) | ≤ 2\*log(n+1) |
| Lookup / Insert / Delete | O(h) | O(log n) |
| Complexity guarantee | Only if balanced | Always guaranteed |
| Use in Java | Simple tree | HashMap buckets, TreeMap |
| Simple Analogy  * Think of a RBT like a **tightrope walker**:   + Red and black colors act like “weights” to prevent it from leaning too far left or right.   + If imbalance occurs, the tree **rotates or recolors** automatically to stay upright. |  |  |

**Binary Search Tree** vs **Binary Search Tree?**

## Key Differences

| **Feature** | **Binary Tree** | **Binary Search Tree (BST)** |
| --- | --- | --- |
| Node order | No order | Left < Parent < Right |
| Search efficiency | O(n) | O(log n) (avg, balanced) |
| Insertion/Deletion | O(n) | O(log n) (avg, balanced) |
| Use cases | Expression trees, hierarchy | Searching, sorted data, maps |
| Traversal result | Any order | Inorder gives sorted sequence |