Depth First Search of a Binary Tree: Java Implementation

Depth First Search (DFS) is a tree traversal algorithm that explores as far as possible along each branch before backtracking. When implemented on a binary tree in Java, DFS can be performed using either recursion or an iterative approach with a stack. Here's an example using both methods:

Recursive DFS Implementation

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class DepthFirstSearch {

    public void dfs(TreeNode node) {
        if (node == null) {
            return;
        }

        // Visit the node (for example, print the value)
        System.out.print(node.val + " ");
```

```
        // Traverse the left subtree
        dfs(node.left);

        // Traverse the right subtree
        dfs(node.right);
    }

    public static void main(String[] args) {
        // Example tree:
        //     1
        //    / \
        //   2   3
        //  / \   \
        // 4   5   6

        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);

        DepthFirstSearch dfs = new DepthFirstSearch();
        dfs.dfs(root); // Output: 1 2 4 5 3 6
    }
}
```

Iterative DFS Implementation

```
import java.util.Stack;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class DepthFirstSearch {

    public void dfs(TreeNode root) {
        if (root == null) {
            return;
        }

        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);

        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
```

```
        // Visit the node (for example, print the value)
        System.out.print(node.val + " ");

        // Push right child to stack first (so that left child is processed first)
        if (node.right != null) {
            stack.push(node.right);
        }

        // Push left child to stack
        if (node.left != null) {
            stack.push(node.left);
        }
    }
}

public static void main(String[] args) {
    // Example tree:
    //     1
    //    / \
    //   2   3
    //  / \   \
    // 4   5   6

    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
```

```
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);

        DepthFirstSearch dfs = new DepthFirstSearch();
        dfs.dfs(root); // Output: 1 2 4 5 3 6
    }
}
```

Explanation

Recursive DFS:

- This method uses the call stack to traverse the tree. It starts at the root, visits the node, and then recursively visits the left and right subtrees.

Iterative DFS:

- This method uses an explicit stack to manage the traversal. The root node is pushed onto the stack, and then the algorithm enters a loop where it pops a node, processes it, and pushes its children (right first, then left) onto the stack. This ensures that the left child is processed before the right child, maintaining the DFS order.

Both methods will visit the nodes in the order: 1, 2, 4, 5, 3, 6 for the example tree.