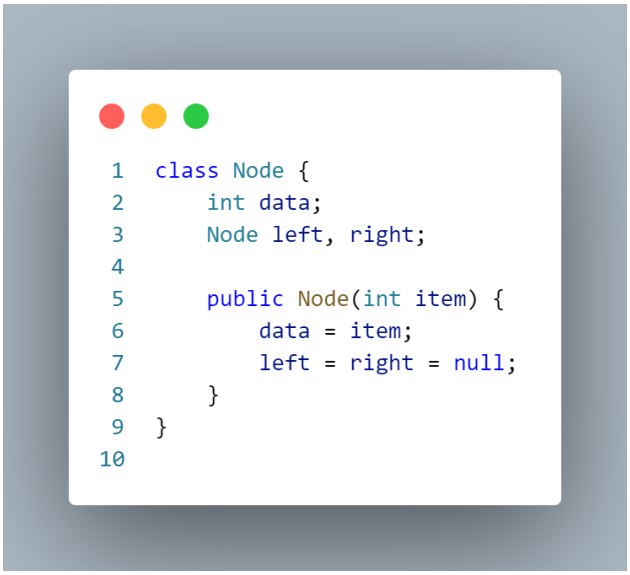
BinarySearch

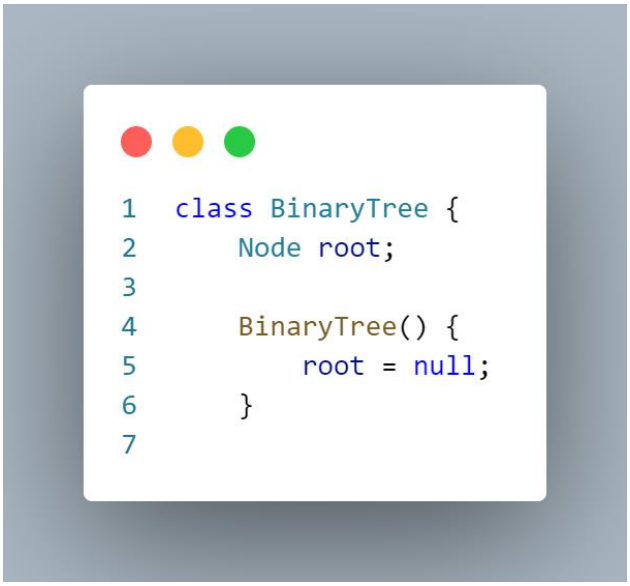
Node Class



```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

```

- Attributes:data: Stores the integer value in the node.
- left and right: References to the left and right child nodes.

BinaryTree Class



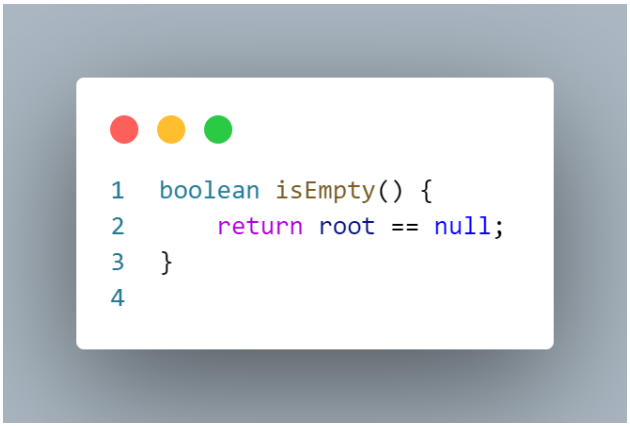
```
class BinaryTree {
    Node root;

    BinaryTree() {
        root = null;
    }

```

- Attributes:root: Represents the root node of the binary tree.

Constructor BinaryTree()

- Purpose: Initializes an empty binary tree with a null root.

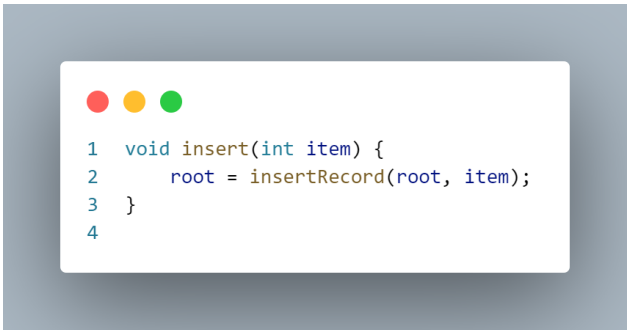
isEmpty()



```
boolean isEmpty() {
    return root == null;
}
```

- Purpose: Checks if the binary tree is empty.
- Returns: true if the root is null (indicating an empty tree), otherwise false.

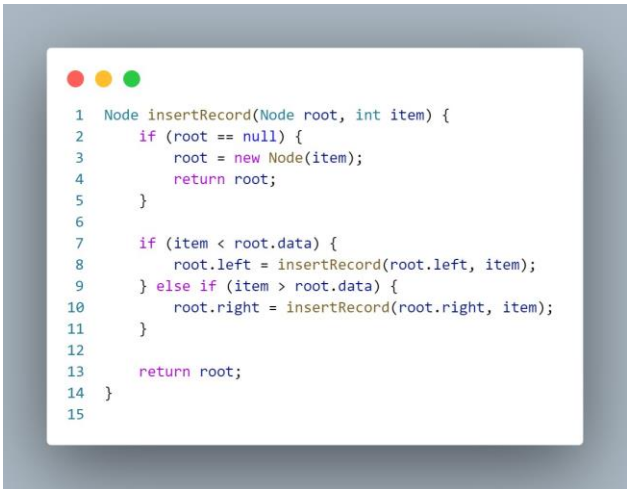
insert(int item)



```
void insert(int item) {
    root = insertRecord(root, item);
}
```

- Purpose: Inserts a node with the given value into the binary tree.
- Parameters: item - Integer value to be inserted into the tree.
- Behavior: Utilizes the insertRecord() method to create and insert the node into the tree based on BST properties.

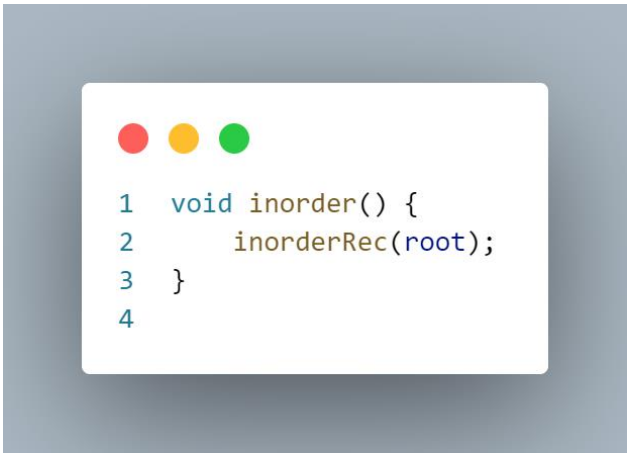
insertRecord(Node root, int item)



```
Node insertRecord(Node root, int item) {
    if (root == null) {
        root = new Node(item);
        return root;
    }

    if (item < root.data) {
        root.left = insertRecord(root.left, item);
    } else if (item > root.data) {
        root.right = insertRecord(root.right, item);
    }

    return root;
}

```

- Purpose: Recursive method to insert a node into the binary tree.
- Parameters: root - Root of the current subtree, item - Integer value to be inserted.
- Returns: The modified root of the subtree.
- Behavior: Traverses the tree recursively to find the appropriate position to insert the node based on its value.

Traversal Methods



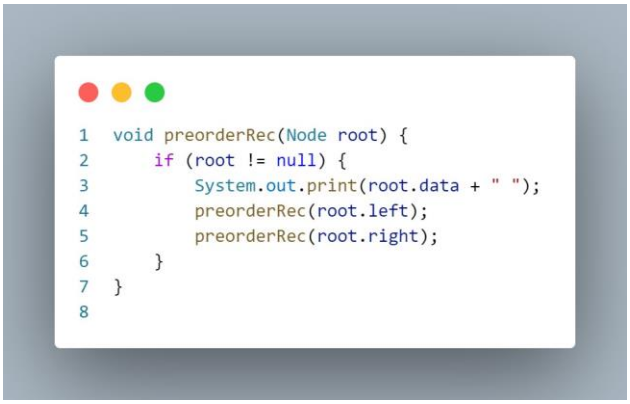
```
void inorder() {
    inorderRec(root);
}

```

```
1       void preorder() {
2           preorderRec(root);
3       }
4
5
```

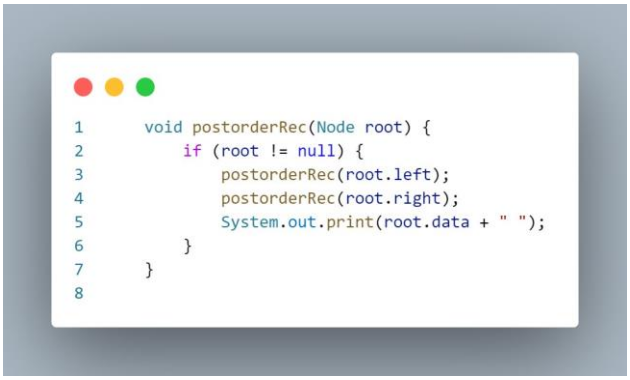
```
1   void postorder() {
2       postorderRec(root);
3   }
4
```

- inorder(), preorder(), postorder(): Initiates the respective traversal by calling their recursive counterparts.

```
1   void inorderRec(Node root) {
2       if (root != null) {
3           inorderRec(root.left);
4           System.out.print(root.data + " ");
5           inorderRec(root.right);
6       }
7   }
8
```



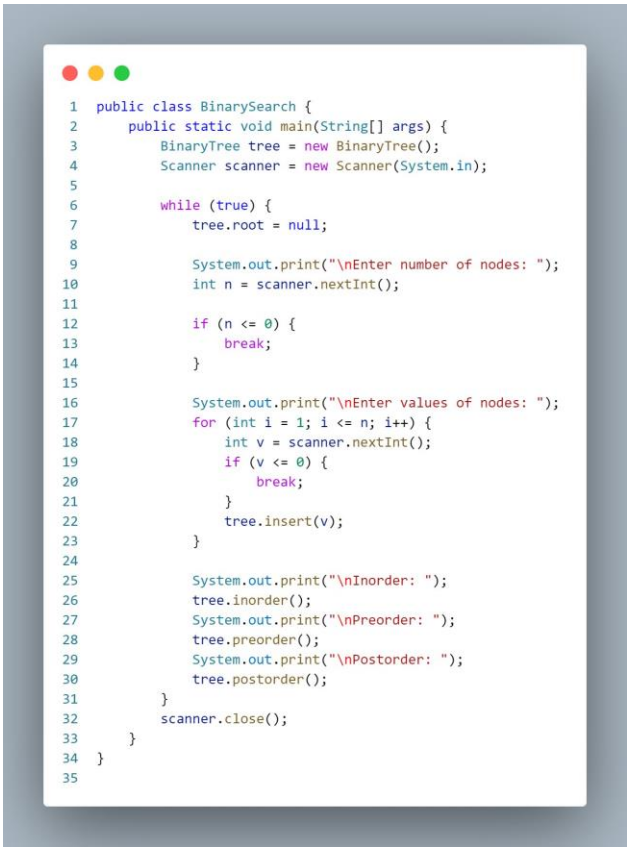
```java
void preorderRec(Node root) {
    if (root != null) {
        System.out.print(root.data + " ");
        preorderRec(root.left);
        preorderRec(root.right);
    }
}
```



```java
    void postorderRec(Node root) {
        if (root != null) {
            postorderRec(root.left);
            postorderRec(root.right);
            System.out.print(root.data + " ");
        }
    }
```

- inorderRec(Node root), preorderRec(Node root), postorderRec(Node root): Recursive methods to perform the respective traversals (inorder, preorder, postorder).

BinarySearch Class (Main)



```java
public class BinarySearch {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            tree.root = null;

            System.out.print("\nEnter number of nodes: ");
            int n = scanner.nextInt();

            if (n <= 0) {
                break;
            }

            System.out.print("\nEnter values of nodes: ");
            for (int i = 1; i <= n; i++) {
                int v = scanner.nextInt();
                if (v <= 0) {
                    break;
                }
                tree.insert(v);
            }

            System.out.print("\nInorder: ");
            tree.inorder();
            System.out.print("\nPreorder: ");
            tree.preorder();
            System.out.print("\nPostorder: ");
            tree.postorder();
        }
        scanner.close();
    }
}
```

- Purpose: Allows user interaction to create and visualize the BST.

main(String[] args)

- Behavior:Creates an instance of BinaryTree.Accepts input for the number of nodes and their values using a Scanner.For each set of values, constructs the BST and displays its inorder, preorder, and postorder traversals.Continues until the user enters a non-positive number of nodes.

The code provides a basic framework to interactively create a BST and view its three different traversals. Users can input values and observe how the tree is structured and traversed accordingly.