# Implementation of a Binary Search Tree (BST) in Java with detailed explanations:

// Class representing a node in the Binary Search Tree class TreeNode

{

int key;

TreeNode left, right;

public TreeNode(int item)

{

key = item;

left = right = null;

}

}

// Class representing the Binary Search Tree public class BinarySearchTree

{

// Root of the Binary Search Tree private TreeNode root;

// Constructor to initialize an empty BST public BinarySearchTree()

{

root = null;

}

// Method to insert a key into the BST public void insert(int key)

{

root = insertRec(root, key);

}

// Recursive method to perform the actual insertion private TreeNode insertRec(TreeNode root, int key)

{

// If the tree is empty, create a new node and return it as the root if (root == null)

{

root = new TreeNode(key); return root;

}

// Otherwise, recur down the tree if (key < root.key)

{

// If the key to be inserted is smaller than the root's key,

// then it belongs to the left subtree root.left = insertRec(root.left, key);

} else if (key > root.key)

{

// If the key to be inserted is larger than the root's key,

// then it belongs to the right subtree root.right = insertRec(root.right, key);

}

// Return the (unchanged) node pointer return root;

}

// Method to perform an in-order traversal of the BST public void inOrderTraversal()

{

inOrderTraversalRec(root);

}

// Recursive method to perform the actual in-order traversal private void inOrderTraversalRec(TreeNode root)

{

if (root != null)

{

// Traverse the left subtree inOrderTraversalRec(root.left);

// Process the current node System.out.print(root.key + " ");

// Traverse the right subtree inOrderTraversalRec(root.right);

}

}

// Method to search for a key in the BST public boolean search(int key)

{

return searchRec(root, key);

}

// Recursive method to perform the actual search

private boolean searchRec(TreeNode root, int key)

{

// Base Cases: root is null or key is present at root if (root == null || root.key == key) {

return root != null; // Return true if key is found, false otherwise

}

// Key is smaller than the root's key if (key < root.key) {

return searchRec(root.left, key);

}

// Key is larger than the root's key return searchRec(root.right, key);

}

// Main method to test the Binary Search Tree public static void main(String[] args)

{

BinarySearchTree bst = new BinarySearchTree();

// Insert keys into the BST bst.insert(50); bst.insert(30); bst.insert(20); bst.insert(40); bst.insert(70); bst.insert(60); bst.insert(80);

// Print in-order traversal

System.out.println("In-order traversal of the BST:"); bst.inOrderTraversal();

System.out.println();

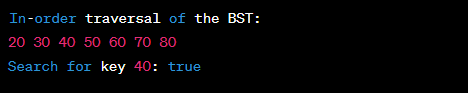
// Search for a key in the BST int searchKey = 40;

System.out.println("Search for key " + searchKey + ": " + bst.search(searchKey));

}

}

# Output:



**Output Explanation:**

In-order traversal of the BST: 20 30 40 50 60 70 80

Search for key 40: true

# Output Explanation:

1. The `inOrderTraversal` method is called, which performs an in-order traversal of the BST and prints the keys in ascending order.

- The output `20 30 40 50 60 70 80` represents the in-order traversal of the BST.

1. The `search` method is called with the key `40`.

- The output `Search for key 40: true` indicates that the key `40` is found in the BST.

# Detailed Code Explanation:

1. **Node Class: `TreeNode`**

* A class to represent a node in the Binary Search Tree (BST).
* Each node has a key (integer value), and left and right child pointers.

class TreeNode { int key;

TreeNode left, right;

public TreeNode(int item) { key = item;

left = right = null;

}

}

# Binary Search Tree Class: `BinarySearchTree`

* A class to represent the Binary Search Tree itself.
* It has a private instance variable `root` representing the root of the tree.

public class BinarySearchTree { private TreeNode root;

// Constructor to initialize an empty BST public BinarySearchTree() {

root = null;

}

# Insertion Method: `insert`

* Method to insert a key into the BST.
* It calls a recursive helper method `insertRec` to perform the actual insertion.

// Method to insert a key into the BST public void insert(int key) {

root = insertRec(root, key);

}

// Recursive method to perform the actual insertion private TreeNode insertRec(TreeNode root, int key) {

if (root == null) {

root = new TreeNode(key); return root;

}

if (key < root.key) {

root.left = insertRec(root.left, key);

} else if (key > root.key) {

root.right = insertRec(root.right, key);

}

return root;

}

# In-Order Traversal: `inOrderTraversal`

* Method to perform an in-order traversal of the BST.
* It calls a recursive helper method `inOrderTraversalRec` to traverse the tree in-order.

// Method to perform an in-order traversal of the BST public void inOrderTraversal() {

inOrderTraversalRec(root);

}

// Recursive method to perform the actual in-order traversal private void inOrderTraversalRec(TreeNode root) {

if (root != null) { inOrderTraversalRec(root.left); System.out.print(root.key + " "); inOrderTraversalRec(root.right);

}

}

# Search Method: `search`

* Method to search for a key in the BST.
* It calls a recursive helper method `searchRec` to perform the actual search.

// Method to search for a key in the BST public boolean search(int key) {

return searchRec(root, key);

}

// Recursive method to perform the actual search private boolean searchRec(TreeNode root, int key) {

if (root == null || root.key == key) { return root != null;

}

if (key < root.key) {

return searchRec(root.left, key);

}

return searchRec(root.right, key);

}

# Main Method: `main`

* A simple `main` method to test the Binary Search Tree.
* It creates a BST, inserts some keys, prints the in-order traversal, and searches for a key.

// Main method to test the Binary Search Tree public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

bst.insert(50); bst.insert(30); bst.insert(20); bst.insert(40); bst.insert(70); bst.insert(60); bst.insert(80);

System.out.println("In-order traversal of the BST:"); bst.inOrderTraversal();

System.out.println();

int searchKey = 40;

System.out.println("Search for key " + searchKey + ": " + bst.search(searchKey));

}

}