

```
1  import java.util.Scanner;
2
3
4  class BTreeNode {
5      BTreeNode left, right;
6      int data;
7
8      public BTreeNode() {
9          left = null;
10         right = null;
11         data = 0;
12     }
13
14     public BTreeNode(int n) {
15         left = null;
16         right = null;
17         data = n;
18     }
19
20
21     public void setLeft(BTreeNode n) {
22         left = n;
23     }
24
25
26     public void setRight(BTreeNode n) {
27         right = n;
28     }
29
30     public BTreeNode getLeft() {
31         return left;
32     }
33
34     public BTreeNode getRight() {
35         return right;
36     }
37
38     public void setData(int d) {
39         data = d;
40     }
41
42     public int getData() {
43         return data;
44     }
45 }
46
```

```

47
48 class BT
49 {
50     private BTreeNode root;
51     public BT( )    {
52         root = null;
53     }
54
55     public boolean isEmpty( )    {
56         return root == null;
57     }
58
59     public void insert(int data)    {
60         root = insert(root, data);
61     }
62
63     private BTreeNode insert(BTreeNode node, int data)    {
64         if (node == null)
65             node = new BTreeNode(data);
66         else    {
67             if (node.getRight() == null)
68                 node.right = insert(node.right, data);
69             else
70                 node.left = insert(node.left, data);
71         }
72         return node;
73     }
74
75     public int countNodes( )    {
76         return countNodes(root);
77     }
78
79     private int countNodes(BTreeNode r)    {
80         if (r == null)
81             return 0;
82         else    {
83             int l = 1;
84             l += countNodes(r.getLeft());
85             l += countNodes(r.getRight());
86             return l;
87         }
88     }
89
90     public boolean search(int val)    {
91         return search(root, val);
92     }
93
94     private boolean search(BTreeNode r, int val)    {

```

```

90  ✓ public boolean search(int val)      {
91  ●   | return search(root, val);
92      | }
93
94  ✓ private boolean search(BTNode r, int val)      {
95      |   if (r.getData() == val)
96      |       | return true;
97      |   if (r.getLeft() != null)
98      |       | if (search(r.getLeft(), val))
99      |           | return true;
100     |   if (r.getRight() != null)
101     |       | if (search(r.getRight(), val))
102     |           | return true;
103     |   return false;
104     | }
105
106  ✓ public void inorder()      {
107     |   inorder(root);
108     | }
109
110  ✓ private void inorder(BTNode r)      {
111  ✓   |   if (r != null)      {
112     |       | inorder(r.getLeft());
113     |       | System.out.print(r.getData() + " ");
114     |       | inorder(r.getRight());
115     |       | }
116     |   }
117
118
119  ✓ public void preorder()      {
120     |   preorder(root);
121     | }
122
123
124  ✓ private void preorder(BTNode r)      {
125  ✓   |   if (r != null)      {
126     |       | System.out.print(r.getData() + " ");
127     |       | preorder(r.getLeft());
128     |       | preorder(r.getRight());
129     |       | }
130     |   }
131
132  ✓ public void postorder()      {
133     |   postorder(root);
134     | }
135

```

```

136 ~ private void postorder(BTNode r)    {
137 ~     if (r != null)    {
138 ~         postorder(r.getLeft());
139 ~         postorder(r.getRight());
140 ~         System.out.print(r.getData() +" ");
141 ~     }
142 ~ }
143 ~ }
144 ~
145 ~ public class BinaryTree {
    Run | Debug
146 ~ public static void main(String[] args)    {
147 ~     Scanner scan = new Scanner(System.in);
148 ~
149 ~
150 ~     BT bt = new BT();
151 ~
152 ~
153 ~     System.out.println(x: "Binary Tree Test\n");
154 ~     char ch;
155 ~
156 ~     do    {
157 ~         System.out.println(x: "\nBinary Tree Operations\n");
158 ~         System.out.println(x: "1. insert ");
159 ~         System.out.println(x: "2. search");
160 ~         System.out.println(x: "3. count nodes");
161 ~         System.out.println(x: "4. check empty");
162 ~
163 ~         int choice = scan.nextInt();
164 ~         switch (choice)    {
165 ~         case 1 :
166 ~             System.out.println(x: "Enter integer element to insert");
167 ~             bt.insert( scan.nextInt() );
168 ~             break;
169 ~
170 ~
171 ~         case 2 :
172 ~             System.out.println(x: "Enter integer element to search");
173 ~             System.out.println("Search result : "+ bt.search( scan.nextInt() ));
174 ~             break;
175 ~         case 3 :
176 ~             System.out.println("Nodes = "+ bt.countNodes());
177 ~             break;
178 ~         case 4 :
179 ~             System.out.println("Empty status = "+ bt.isEmpty());
180 ~             break;
181 ~         default :
182 ~             System.out.println(x: "Wrong Entry \n ");

```

```

162
163     int choice = scan.nextInt();
164     switch (choice)    {
165     case 1 :
166         System.out.println(x: "Enter integer element to insert");
167         bt.insert( scan.nextInt() );
168         break;
169
170
171     case 2 :
172         System.out.println(x: "Enter integer element to search");
173         System.out.println("Search result : "+ bt.search( scan.nextInt() ));
174         break;
175
176     case 3 :
177         System.out.println("Nodes = "+ bt.countNodes());
178         break;
179
180     case 4 :
181         System.out.println("Empty status = "+ bt.isEmpty());
182         break;
183
184     default :
185         System.out.println(x: "Wrong Entry \n ");
186         break;
187     }
188
189     System.out.print(s: "\nPost order : ");
190     bt.postorder();
191     System.out.print(s: "\nPre order : ");
192     bt.preorder();
193     System.out.print(s: "\nIn order : ");
194     bt.inorder();
195     System.out.println(x: "\n\nDo you want to continue (Type y or n) \n");
196     ch = scan.next().charAt(index: 0);
197     } while (ch == 'Y' || ch == 'y');
198 }
199 }

```

Binary Tree Operations

1. insert
2. search
3. count nodes
4. check empty

1

Enter integer element to insert

5

Post order : 5

Pre order : 5

In order : 5

Do you want to continue (Type y or n)

y

Binary Tree Operations

1. insert
2. search
3. count nodes
4. check empty

1

Enter integer element to insert

4

Post order : 4 5

Pre order : 5 4

In order : 5 4

Do you want to continue (Type y or n)

y

Binary Tree Operations

1. insert
2. search
3. count nodes
4. check empty

1

Enter integer element to insert

8

Post order : 8 4 5

Pre order : 5 8 4

In order : 8 5 4

Do you want to continue (Type y or n)

n

PS C:\Users\Administrator> █