```
1    public class BinarySearchTree
2    {
3        private TreeNode root;
4
5        public BinarySearchTree ()
6        {
7            root  = null;
8        }
9
10       public void insert(Comparable obj)
11       {
12           TreeNode newNode = new TreeNode(obj, null, null);
13           if (root == null)
14               root  = newNode;
15           else
16               insertNode (root, newNode );
17       }
18
19       public void insertNode (TreeNode current, TreeNode newNode )
20       {
21           Comparable newValue = (Comparable ) newNode.getValue ();
22           Comparable currValue = (Comparable ) current.getValue ();
23
24           if (newValue.compareTo (currValue ) < 0)
25           {
26               if (current.getLeft () == null)
27                   current. setLeft (newNode );
28               else
29                   insertNode (current.getLeft (), newNode );
30           }
31           else
32           {
33               if (current.getRight () == null)
34                   current. setRight (newNode );
35               else
36                   insertNode (current.getRight (), newNode );
37           }
38       }
39
40       public void print ()
41       {
42           if (root != null)
43               printNodes (root);
44       }
45
46       public void printNodes (TreeNode current )
47       {
48           if (current != null)
49           {
50               printNodes (current.getLeft ());
51               System .out.println (current.getValue ());
52               printNodes (current.getRight ());
53           }
54       }

55
56       public boolean find(Comparable key)
57       {
58           if (root == null)
59               return false;
60           else
61               return findNode (root, key );
62       }
63
64       public boolean findNode (TreeNode current, Comparable key)
65       {
66           if (current == null)
```

```
67                  return false;
68              else
69              {
70                  Comparable currValue = (Comparable) current.getValue();
71                  if (key.compareTo(currValue) == 0)
72                      return true;
73                  else if (key.compareTo(currValue) < 0)
74                      return findNode(current.getLeft(), key);
75                  else
76                      return findNode(current.getRight(), key);
77              }
78          }
79
80          public void printPostorder()
81          {
82              if (root != null)
83                  printNodesPostorder(root);
84          }
85
86          public void printNodesPostorder(TreeNode current)
87          {
88              if (current != null)
89              {
90                  printNodes(current.getLeft());
91                  printNodes(current.getRight());
92                  System.out.println(current.getValue());
93              }
94          }
95
96          public int countNodesHelper()
97          {
98              if (root == null)
99                  return 0;
100             return countNodes(root);
101         }
102
103         public  int countNodes(TreeNode root)
104         {
105             if (root == null)
106                 return 0;
107
108             return 1 + countNodes(root.getRight()) + countNodes(root.getLeft())
109         }
110
111         // ************************************************
112         // findMin only works if the tree stores Integer objects
113
114         public Object findMinValueHelper()
115         {
116             if (root == null)
117                 return 0;
118
119             return findMinValue(root);
120         }
121
122         public Object findMinValue(TreeNode root)
123         {
124             if (root.getLeft() == null)
125                 return root.getValue();
126
127             return findMinValue(root.getLeft());
128         }
129
130         // ***************************************
131         // treeSum only works if the tree stores Integer objects
132
```

```
133         public int treeSumHelper ()
134         {
135             if (root == null)
136                 return 0;
137 
138             return treeSum (root);
139         }
140 
141         public static int treeSum (TreeNode root)
142         {
143             if (root == null)
144                 return 0;
145 
146             return ((Integer) root.getValue ()).intValue () +
      treeSum (root.getLeft ()) + treeSum (root.getRight ());
147         }
148 
149         // ************************************************
150 
151         public int countLeafsHelper ()
152         {
153             if (root == null)
154                 return 0;
155 
156             return countLeafs (root);
157         }
158 
159         public int countLeafs (TreeNode root)
160         {
161             if (root == null)
162                 return 0;
163             else if (root.getLeft () == null && root.getRight () == null)
164                 return 1;
165 
166             return countLeafs (root.getLeft ()) + countLeafs (root.getRight ());
167         }
168 
169         // ***************************************
170 
171         public int treeDepthHelper ()
172         {
173             if (root == null)
174                 return 0;
175 
176             return treeDepth (root);
177         }
178 
179         public  int treeDepth (TreeNode root)
180         {
181             int depth = 0;
182 
183             if (root == null)
184                 return 0;
185 
186             if (root.getRight () == null && root.getLeft () == null)
187                 return depth;
188 
189             if (treeDepth (root.getRight ()) > treeDepth (root.getLeft ()))
190                 return depth + treeDepth (root.getRight ()) + 1;
191 
192             return depth + treeDepth (root.getLeft ()) + 1;
193         }
194 
195         // ****************************************
196 
197         public int internalPathLengthHelper ()
198         {
```

```
199             if (root == null)
200                 return 0;
201
202             return internalPathLength (root, 0);
203         }
204
205         public int internalPathLength (TreeNode root, int pathsSoFar )
206         {
207             if (root == null)
208                 return 0;
209             else
210                 return pathsSoFar +
211                             internalPathLength (root.getLeft (), pathsSoFar + 1)
212                             internalPathLength (root.getRight (), pathsSoFar + 1)
213         }
214
215         // ****************************************
216
217         public static boolean areSimilar (TreeNode tree1, TreeNode tree2 )
218         {
219             //  two trees are similar if they have the same exact shape and
    structure
220
221             return  true;
222         }
223
224         // ****************************************
225
226         public int distance (TreeNode node1, TreeNode node2 )
227         {
228             return 0;
229         }
230     }
231
```