

```
2 // BinarySearchTree.java          by Dale/Joyce/Weems          Chapter 8
6
7 package bst_Package;
8
9 import org.w3c.dom.Node;
13
14 public class BinarySearchTree<T extends Comparable<T>>
15         implements BSTInterface<T>
16 {
17     protected BSTNode<T> root;      // reference to the root of this BST
18
19     boolean found;    // used by remove
20
21     // for traversals
22     protected LinkedUnbndQueue<T> inOrderQueue;    // queue of info
23     protected LinkedUnbndQueue<T> preOrderQueue;  // queue of info
24     protected LinkedUnbndQueue<T> postOrderQueue; // queue of info
25
26     public BinarySearchTree()
27     // Creates an empty BST object.
28     {
29         root = null;
30     }
31
32     // Recursively count each leaf node on BST
33     // Returns count of leaves
34     private int recLeafCount(BSTNode<T> node) {
35         int count = 0;
36         if (node == null)
37             count += 0;
38         else if (node.getLeft() == null && node.getRight() == null)
39             count = 1;
40         else
41             count += recLeafCount(node.getLeft()) + recLeafCount(node.getRight());
42
43         return count;
44     }
45
46     // Count the leaf nodes on the Binary Search Tree
47     public int leafCount() {
48         return recLeafCount(root);
49     }
50
51     // Count the single parents on the Binary Search Tree
52     public int singleParentCount() {
53         return recSingleParentCount(root);
54     }
55
56     // Recursively count all the single the parents on the Binary Search tree
57     // Returns count of single parents
58     private int recSingleParentCount(BSTNode<T> node) {
59         if (node == null)
60             return 0;
61         else if ((node.getLeft() == null && node.getRight() != null) ||
62                 (node.getLeft() != null && node.getRight() == null))
63             return 1;
64         else
```

```
65         return recSingleParentCount(node.left) + recSingleParentCount(node.right);
66     }
67
68     public boolean isEmpty()
69
70
71
72
73
74     private int recSize(BSTNode<T> tree)
75
76
77
78
79
80
81
82
83     public int size()
84
85
86
87
88
89     public int size2()
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112     private boolean recContains(T element, BSTNode<T> tree)
113
114
115
116
117
118
119
120
121
122
123
124
125
126     public boolean contains (T element)
127
128
129
130
131
132
133     private T recGet(T element, BSTNode<T> tree)
134
135
136
137
138
139
140
141
142
143
144
145
146
147     public T get(T element)
148
149
150
151
152
153
154
155     private BSTNode<T> recAdd(T element, BSTNode<T> tree)
156
157
158
159
160
161
162
163
164
165
166
167
168     public void add (T element)
169
170
171
172
173
174     private T getPredecessor(BSTNode<T> tree)
175
176
177
178
179
180
181
182     private BSTNode<T> removeNode(BSTNode<T> tree)
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205     private BSTNode<T> recRemove(T element, BSTNode<T> tree)
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223     public boolean remove (T element)
224
225
226
227
228
229
230
231     private void inOrder(BSTNode<T> tree)
232
233
234
235
236
237
238
239
240
241
242     private void preOrder(BSTNode<T> tree)
243
244
245
246
247
248
249
250
251
252
253     private void postOrder(BSTNode<T> tree)
254
255
256
257
258
259
260
261
262
263
264     public int reset(int orderType)
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279     public T getNext (int orderType)
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309     public void showStructure ( )
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325     private void showSub ( BSTNode p, int level )
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354 }
```