

```
1 import java.util.Scanner;
2 public class Main
3 {
4     public static void main(String[] args){
5         Scanner keyboard = new Scanner(System.in);
6         StuData sd = new StuData();
7
8
9         displayMenu();
10        int option = keyboard.nextInt();
11
12        while(option != 8){
13            if (option == 1){
14                String[] parms = getParm("stuData").split("\\s+");
15                sd.insert(Integer.valueOf(parms[0].trim()),
16                        parms[1],
17                        Integer.valueOf(parms[2].trim()),
18                        Double.valueOf(parms[3].trim()));
19            }
20            else if (option == 2){sd.delete(Integer.valueOf(getParm("id").t
21            else if (option == 3){sd.search(Integer.valueOf(getParm("id").t
22            else if (option == 4){
23                String[] parms = getParm("stuData").split("\\s+");
24                sd.modify(Integer.valueOf(parms[0].trim()),
25                        parms[1],
26                        Integer.valueOf(parms[2].trim()),
27                        Double.valueOf(parms[3].trim()));
28            }
29            else if (option == 5){sd.display();}
30            else if (option == 6){sd.upload(getParm("path"));}
31            else if (option == 6){sd.download(getParm("path"));}
32            else{System.out.println("Invalid Option");}
33
34            displayMenu();
35            option = keyboard.nextInt();
36        }
37
38
39    }
40    private static void displayMenu(){
41        System.out.println("1 - Add a student\n" +
42                "2 - Delete a student\n" +
43                "3 - Search for a student\n" +
44                "4 - Modify a student\n" +
45                "5 - Display all students\n" +
46                "6 - Upload a data file\n" +
47                "7 - Download the data file\n" +
48                "8 - Exit\n" +
```

```
49         "Choose an option - ");
50     }
51
52
53     private static String getParm(String parm){
54         Scanner keyboard = new Scanner(System.in);
55         String toReturn = "";
56         if(parm == "path"){
57             System.out.println("Enter the file path: ");
58             toReturn = keyboard.nextLine();
59         }
60         else if (parm == "stuData"){
61             System.out.println("Enter student id, name, age, and gpa sepeara");
62             toReturn = keyboard.nextLine();
63         }
64         else if (parm == "id"){
65             System.out.println("Enter student ID #: ");
66             toReturn = String.valueOf(keyboard.nextInt());
67         }
68         else{
69             System.out.println("Weird Error, you better contact IT");
70         }
71         return toReturn;
72     }
73 }
74
```

```
1 import java.io.FileReader;
2 import java.io.File;
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 public class StuData
9 {
10     private class node{
11         int id;
12         String lastName;
13         int age;
14         double gpa;
15         node left;
16         node right;
17
18         private node(int id, String lastName, int age, double gpa, node left, node right){
19             this.id = id;
20             this.lastName = lastName;
21             this.age = age;
22             this.gpa = gpa;
23             this.left = left;
24             this.right = right;
25         }
26     }
27
28     node root;
29
30     public StuData()
31     {
32         root = null;
33     }
34
35     public boolean insert(int id, String lastName, int age, double gpa){
36         if(age < 10 || age > 100){
37             System.out.println("Invalid age");
38             return false;
39         }
40         else if(id < 1 || id > 10000){
41             System.out.println("Invalid ID");
42             return false;
43         }
44         else if(gpa < 0.0 || gpa > 4.0){
45             System.out.println("Invalid GPA");
46             return false;
47         }
48         else if(root == null){//special case, empty tree
```

```
49         root = new node(id, lastName, age, gpa, null, null);
50         return true;
51     }
52     else{
53         node temp = root;
54         node prev = null;
55         while(temp != null){
56             if(id == temp.id){ //student already exists
57                 System.out.println("Student already exists");
58                 return false;
59             }
60             else if(id < temp.id){ //go left
61                 prev = temp;
62                 temp = temp.left;
63             }
64             else{ //go right
65                 prev = temp;
66                 temp = temp.right;
67             }
68         }
69         if(id < prev.id){
70             prev.left = new node(id, lastName, age, gpa, null, null);
71             return true;
72         }
73         else{
74             prev.right = new node(id, lastName, age, gpa, null, null);
75             return true;
76         }
77     }
78 }
79
80 public boolean delete(int id){
81     node prev = getPrev(id);
82     if(prev == null){//delete root
83         delete(root, null);
84         return true;
85     }
86     else if(prev.left == null && prev.right == null){// id didn't exist
87         return false;
88     }
89     else if(prev.left != null && prev.left.id == id){// delete left node
90         delete(prev.left, prev);
91         return true;
92     }
93     else if(prev.right != null && prev.right.id == id){ //delete right
94         delete(prev.right, prev);
95         return true;
96     }
97 }
```

```
97         else{
98             return false;
99         }
100     }
101
102     public boolean search(int id){
103         node prev = getPrev(id);
104         if(prev == null){//print root
105             print(root);
106             return true;
107         }
108         else if(prev.left == null && prev.right == null){// id didn't exist
109             System.out.println("Student not found");
110             return false;
111         }
112         else if(prev.left != null && prev.left.id == id){// print left node
113             print(prev.left);
114             return true;
115         }
116         else if(prev.right != null && prev.right.id == id){ //print right r
117             print(prev.right);
118             return true;
119         }
120         else{
121             System.out.println("Student not found");
122             return false;
123         }
124     }
125
126     public boolean modify(int id, String lastName, int age, double gpa){
127         node prev = getPrev(id);
128         if(prev == null){//modify root
129             modify(root, lastName, age, gpa);
130             return true;
131         }
132         else if(prev.left == null && prev.right == null){// id didn't exist
133             System.out.println("Student not found");
134             return false;
135         }
136         else if(prev.left != null && prev.left.id == id){// print left node
137             modify(prev.left, lastName, age, gpa);
138             return true;
139         }
140         else if(prev.right != null && prev.right.id == id){ //print right r
141             modify(prev.right, lastName, age, gpa);
142             return true;
143         }
144         else{
```

```
145         System.out.println("Student not found");
146         return false;
147     }
148 }
149
150 public void display(){
151     display(root);
152 }
153
154 public void upload(String path){
155     System.out.println("Upload");
156     try{ //grab inputs from file
157         FileReader fileReader = new FileReader(new File(path));
158         BufferedReader br = new BufferedReader(fileReader);
159
160         System.out.println("Setup success");
161
162         root = null; // delete current tree
163         String line;
164         while((line=br.readLine())!=null){
165             //do sstuff
166             System.out.println("Lineread");
167             int id = Integer.valueOf(line.substring(4,9).trim());
168             System.out.println("ID: " + id);
169             int age = Integer.valueOf(line.substring(17,19).trim());
170             System.out.println("AGE: " + age);
171             double gpa = Double.valueOf(line.substring(26,32));
172             System.out.println("GPA: " + gpa);
173             String lastName = line.substring(44);
174         }
175     }
176     catch (Exception e) {
177         System.out.println(e.getClass());
178     }
179 }
180 public void download(String path){
181     try{
182         // Create file
183         FileWriter fstream = new FileWriter(path);
184         BufferedWriter out = new BufferedWriter(fstream);
185         out.write(downloadFormat(root));
186         //Close the output stream
187         out.close();
188     }catch (Exception e){//Catch exception if any
189         System.err.println("Error: " + e.getMessage());
190     }
191 }
192 }
```

```
193
194     private String downloadFormat(node temp){
195         if(temp == null){
196             return "";
197         }
198         else{
199             return format(temp) + downloadFormat(temp.left) + downloadForma
200         }
201     }
202
203     private void display(node temp){
204         if(temp == null){
205             return;
206         }
207         else{
208             display(temp.left);
209             print(temp);
210             display(temp.right);
211         }
212     }
213
214     private void modify(node temp, String lastName, int age, double gpa){
215         temp.lastName = lastName;
216         temp.age = age;
217         temp.gpa = gpa;
218     }
219
220     private String format(node temp){
221         return String.format("ID: %5d  AGE: %3d  GPA: %1.3f  LASTNAME: %S %
222     }
223
224
225
226 }
227
228     private void print(node temp){
229         System.out.print(format(temp));
230     }
231
232     private boolean delete(node temp, node prev){
233         if(temp.left == null){ //no left subtree
234             System.out.println("No left subtree");
235             if (temp == root){ //deleting root
236                 root = temp.right;
237             }
238             else if (temp == prev.left){
239                 prev.left = temp.right;
240             }

```

```
241         else{
242             prev.left = temp.right;
243         }
244     }
245     else if(temp.right == null){//no right subtree
246         System.out.println("No right subtree");
247         if (temp == root){ //deleting root
248             root = temp.left;
249         }
250         else if (temp == prev.right){
251             prev.right = temp.left;
252         }
253         else{
254             prev.left = temp.left;
255         }
256     }
257     else{//left and right subtrees present
258         System.out.println("both subtree");
259         node p = temp.left;
260         node q = temp;
261         while(p.right != null){ //get max of left subtree
262             q = p;
263             p = p.right;
264         }
265         temp.id = p.id;
266         temp.age = p.age;
267         temp.gpa = p.gpa;
268         temp.lastName = p.lastName;
269         if(q == p.left){
270             q.left = p.left;
271         }
272         else{
273             q.right = p.left;
274         }
275     }
276     return true;
277 }
278
279 private node getPrev(int id){
280     node temp = root;
281     node prev = null;
282     while(temp != null){
283         if(id == temp.id){ // found it
284             break;
285         }
286         else if(id < temp.id){ //go left
287             prev = temp;
288             temp = temp.left;
```



```
289         }
290         else{ //go right
291             prev = temp;
292             temp = temp.right;
293         }
294     }
295     return prev;
296 }
297
298 }
299
```

```
1 import java.util.Scanner;
2 import java.util.Random;
3 public class Main
4 {
5     public static void main(String[] args){
6         Scanner keyboard = new Scanner(System.in);
7
8         displayMenu();
9         int option = keyboard.nextInt();
10        bst bs = null;
11        long seed;
12
13        while(option != 4){
14            if (option == 1){
15                System.out.println("Enter a the desired number of nodes");
16                int n = keyboard.nextInt();
17                System.out.println("Enter an RNG seed");
18                seed = keyboard.nextLong();
19                bs = new bst(n,seed);
20            }
21            else if (option == 2){
22                if(bs != null){
23                    System.out.println("Height: " + bs.height());
24                    System.out.println("Count: " + bs.count());
25                }
26                else{
27                    System.out.println("You havn't initialized your tree ye
28                }
29            }
30            else if (option == 3){
31                if(bs != null){
32                    System.out.println("Enter an RNG seed");
33                    seed = keyboard.nextLong();
34
35                    Random rand = new Random(seed);
36
37                    long start = System.currentTimeMillis();
38                    System.out.println("Start: " + start);
39                    for(int i=0;i<1000000;i++){
40                        bs.search((rand.nextInt(1000000000)+1));
41                    }
42                    long stop = System.currentTimeMillis();
43                    System.out.println("Stop: " + stop);
44                    double time = Double.valueOf(stop-start);
45                    System.out.println("Total time required: " + time);
46                }
47                else{
48                    System.out.println("You havn't initialized your tree ye
```

```
49         }
50     }
51     else{System.out.println("Invalid Option");}
52
53     displayMenu();
54     option = keyboard.nextInt();
55 }
56
57
58 }
59 private static void displayMenu(){
60     System.out.println("1 - Create BST\n" +
61         "2 - Display height & # of nodes\n" +
62         "3 - Perform 10^6 random searches\n" +
63         "4 - Exit\n" +
64         "Choose an option - ");
65 }
66 }
67
68
```

```
1 import java.util.Random;
2 public class bst
3 {
4     private class node{
5         int data;
6         node left;
7         node right;
8         node parent;
9
10        private node(int data, node left, node right, node parent){
11            this.data = data;
12            this.left = left;
13            this.right = right;
14            this.parent = parent;
15        }
16    }
17
18    node root;
19    //needs: insert, constructor using rng, height, count nodes
20
21    public bst(){
22        this.root = null;
23    }
24    public bst(int n, long seed){
25        this.root = null;
26        Random rand = new Random(seed);
27        for(int i=0;i<n;i++){
28            this.insert((rand.nextInt(1000000000)+1));
29        }
30    }
31
32    public int count(){
33        return count(root);
34    }
35
36    public int height(){
37        return height(root);
38    }
39
40    public boolean insert(int data){
41        if(root == null){ //special case, empty tree
42            root = new node(data, null, null, null);
43            return true;
44        }
45        else{
46            node temp = root;
47            node prev = null;
48            while(temp != null){
```

```
49         if(temp.data == data){
50             return false;
51         }
52         else if(data > temp.data){ //go right
53             prev = temp;
54             temp = temp.right;
55         }
56         else{ //go left
57             prev = temp;
58             temp = temp.left;
59         }
60     }
61     if(data > prev.data){ //create right
62         prev.right = new node(data, null, null, prev);
63         return true;
64     }
65     else{ //create left
66         prev.left = new node(data, null, null, prev);
67         return true;
68     }
69 }
70 }
71
72 public boolean search(int data){
73     node temp = root;
74     while(temp != null){
75         if(temp.data == data){
76             return true;
77         }
78         else if(data > temp.data){ //go right
79             temp = temp.right;
80         }
81         else{ //go left
82             temp = temp.left;
83         }
84     }
85     return false;
86 }
87
88 private int count(node temp){
89     if(temp == null){
90         return 0;
91     }
92     else{
93         return 1 + count(temp.left) + count(temp.right);
94     }
95 }
96 }
```

```
97     private int height(node temp){
98         if(temp == null){
99             return 0;
100        }
101        int lheight = height(temp.left);
102        int rheight = height(temp.right);
103        if(lheight > rheight){
104            return 1 + lheight;
105        }
106        else{
107            return 1 + rheight;
108        }
109    }
110 }
111
```

```
1 import java.util.Scanner;
2 import java.util.Random;
3 public class Main
4 {
5     public static void main(String[] args){
6         Scanner keyboard = new Scanner(System.in);
7
8         displayMenu();
9         int option = keyboard.nextInt();
10        avl bs = null;
11        long seed;
12
13        while(option != 4){
14            if (option == 1){
15                System.out.println("Enter a the desired number of nodes");
16                int n = keyboard.nextInt();
17                System.out.println("Enter an RNG seed");
18                seed = keyboard.nextLong();
19                bs = new avl(n,seed);
20            }
21            else if (option == 2){
22                if(bs != null){
23                    System.out.println("Height: " + bs.height());
24                    System.out.println("Count: " + bs.count());
25                }
26                else{
27                    System.out.println("You havn't initialized your tree ye
28                }
29            }
30            else if (option == 3){
31                if(bs != null){
32                    System.out.println("Enter an RNG seed");
33                    seed = keyboard.nextLong();
34
35                    Random rand = new Random(seed);
36
37                    long start = System.currentTimeMillis();
38                    System.out.println("Start: " + start);
39                    for(int i=0;i<1000000;i++){
40                        bs.search((rand.nextInt(1000000000)+1));
41                    }
42                    long stop = System.currentTimeMillis();
43                    System.out.println("Stop: " + stop);
44                    double time = Double.valueOf(stop-start);
45                    System.out.println("Total time required: " + time);
46                }
47                else{
48                    System.out.println("You havn't initialized your tree ye
```

```
49         }
50     }
51     else{System.out.println("Invalid Option");}
52
53     displayMenu();
54     option = keyboard.nextInt();
55 }
56
57
58 }
59 private static void displayMenu(){
60     System.out.println("1 - Create AVL tree\n" +
61         "2 - Display height & # of nodes\n" +
62         "3 - Perform 10^6 random searches\n" +
63         "4 - Exit\n" +
64         "Choose an option - ");
65 }
66 }
67
68
```



```
1 import java.util.Random;
2 public class avl
3 {
4     private class node{
5         int data;
6         int height;
7         node left;
8         node right;
9         node parent;
10
11         private node(int data, int height, node left, node right, node parent)
12         {
13             this.data = data;
14             this.height = height;
15             this.left = left;
16             this.right = right;
17             this.parent = parent;
18         }
19     }
20     node root;
21     //needs: insert, constructor using rng, height, count nodes
22
23     public avl(){
24         this.root = null;
25     }
26     public avl(int n, long seed){
27         this.root = null;
28         Random rand = new Random(seed);
29         for(int i=0;i<n;i++){
30             //System.out.println("Insert # " + i);
31             this.insert((rand.nextInt(1000000000)+1));
32         }
33     }
34
35     public int count(){
36         return count(root);
37     }
38
39     public int height(){
40         return root.height;
41     }
42
43     public boolean insert(int data){
44         if(root == null){ //special case, empty tree
45             root = new node(data, 1, null, null, null);
46             return true;
47         }
48         else{
```

```
49         node temp = root;
50         node prev = null;
51         //System.out.println("About to search for: " + data);
52         while(temp != null){
53             if(temp.data == data){
54                 return false;
55             }
56             else if(data > temp.data){ //go right
57                 prev = temp;
58                 temp = temp.right;
59             }
60             else{ //go left
61                 prev = temp;
62                 temp = temp.left;
63             }
64         }
65         System.out.println("Got temp");
66         if(data > prev.data){ //create right
67             prev.right = new node(data, 1, null, null, prev);
68             updateHeight(prev.right);
69             return true;
70         }
71         else{ //create left
72             prev.left = new node(data, 1, null, null, prev);
73             updateHeight(prev.left);
74             return true;
75         }
76     }
77 }
78
79
80 public boolean search(int data){
81     node temp = root;
82     while(temp != null){
83         if(temp.data == data){
84             return true;
85         }
86         else if(data > temp.data){ //go right
87             temp = temp.right;
88         }
89         else{ //go left
90             temp = temp.left;
91         }
92     }
93     return false;
94 }
95 public void prePrint(){
96     prePrint(root);
97 }
```

```
97     }
98
99     private void prePrint(node temp){
100         if(temp == null){
101             return;
102         }
103         else{
104             System.out.println(temp.data);
105             prePrint(temp.left);
106             System.out.println("_____");
107             prePrint(temp.right);
108         }
109     }
110
111     private void updateHeight(node temp){
112         if(temp == null){// at root, we're done
113             return;
114         }
115         if(height(temp.left)==temp.height || height(temp.right)==temp.height){
116             temp.height = temp.height+1;
117         }
118         if(imbalanced(temp)){
119             rebalance(temp);
120         }
121         updateHeight(temp.parent);
122     }
123
124     private int height(node temp){
125         if(temp == null){
126             return 0;
127         }
128         else{
129             return temp.height;
130         }
131     }
132
133     private boolean imbalanced(node temp){
134         int lheight = height(temp.left);
135         int rheight = height(temp.right);
136         if(rheight - lheight < -1 || rheight - lheight > 1){
137             return true;
138         }
139         else{
140             return false;
141         }
142     }
143
144     private void rebalance(node temp){
```

```
145         if(height(temp.left) > height(temp.right)){ //left heavy
146             if(height(temp.left.left) > height(temp.left.right)){ //left he
147                 node parent = temp.parent;
148                 node nodeA = temp;
149                 node nodeB = temp.left;
150                 node subY = temp.left.right;
151
152                 nodeA.left = subY;
153                 nodeA.parent = nodeB;
154                 nodeB.right = nodeA;
155                 nodeB.parent = parent;
156                 if(subY != null){subY.parent = nodeA;}
157                 if(parent == null){ //special case, temp is root
158                     root = nodeB;
159                 }
160                 else if(parent.left == temp){
161                     parent.left = nodeB;
162                 }
163                 else{
164                     parent.right = nodeB;
165                 }
166                 //update heights
167                 temp.height = Math.max(height(temp.left),height(temp.right))
168                 temp = nodeB;
169                 temp.height = Math.max(height(temp.left),height(temp.right))
170             }
171         else{//left heavy inside
172             node parent = temp.parent;
173             node nodeA = temp;
174             node nodeB = temp.left;
175             node nodeC = temp.left.right;
176             node subY = temp.left.right.left;
177             node subZ = temp.left.right.right;
178
179             if(parent == null){ //special case, temp is root
180                 root = nodeB;
181             }
182             else if(parent.left == temp){
183                 parent.left = nodeC;
184             }
185             else{
186                 parent.right = nodeC;
187             }
188             nodeA.left = subZ;
189             nodeA.parent = nodeC;
190             nodeB.right = subY;
191             nodeB.parent = nodeC;
192             nodeC.left = nodeB;
```

```
193         nodeC.right = nodeA;
194         nodeC.parent = parent;
195         if(subY != null){subY.parent = nodeB;}
196         if(subZ != null){subZ.parent = nodeA;}
197         //update heights
198         temp.height = Math.max(height(temp.left),height(temp.right))
199         temp = nodeB;
200         temp.height = Math.max(height(temp.left),height(temp.right))
201         temp = nodeC;
202         temp.height = Math.max(height(temp.left),height(temp.right))
203     }
204 }
205 else{ //right heavy
206     if(height(temp.right.left) > height(temp.right.right)){//right
207         node parent = temp.parent;
208         node nodeA = temp;
209         node nodeB = temp.right;
210         node nodeC = temp.right.left;
211         node subY = temp.right.left.left;
212         node subZ = temp.right.left.right;
213
214         if(parent == null){ //special case, temp is root
215             root = nodeC;
216         }
217         else if(parent.left == temp){
218             parent.left = nodeC;
219         }
220         else{
221             parent.right = nodeC;
222         }
223         nodeA.right = subY;
224         nodeA.parent = nodeC;
225         nodeB.left = subZ;
226         nodeB.parent = nodeC;
227         nodeC.left = nodeA;
228         nodeC.right = nodeB;
229         nodeC.parent = parent;
230         if(subY != null){subY.parent = nodeA;}
231         if(subZ != null){subZ.parent = nodeB;}
232         //update heights
233         temp.height = Math.max(height(temp.left),height(temp.right))
234         temp = nodeB;
235         temp.height = Math.max(height(temp.left),height(temp.right))
236         temp = nodeC;
237         temp.height = Math.max(height(temp.left),height(temp.right))
238     }
239     else{//right heavy outside
240         node parent = temp.parent;
```

```
241         node nodeA = temp;
242         node nodeB = temp.right;
243         node subY = temp.right.left;
244
245         nodeA.right = subY;
246         nodeA.parent = nodeB;
247         nodeB.left = nodeA;
248         nodeB.parent = parent;
249         if(subY != null){subY.parent = nodeA;}
250
251         if(parent == null){ //special case, temp is root
252             root = nodeB;
253         }
254         else if(parent.left == temp){
255             parent.left = nodeB;
256         }
257         else{
258             parent.right = nodeB;
259         }
260         //update heights
261         temp.height = Math.max(height(temp.left),height(temp.right))
262         temp = nodeB;
263         temp.height = Math.max(height(temp.left),height(temp.right))
264     }
265 }
266
267
268 private int count(node temp){
269     if(temp == null){
270         return 0;
271     }
272     else if(temp.right == null && temp.left == null){
273         return 1;
274     }
275     else{
276         return 1 + count(temp.left) + count(temp.right);
277     }
278 }
279 }
280
```

```
1 import java.util.Scanner;
2 public class Main
3 {
4     public static void main(String[] args){
5         Scanner keyboard = new Scanner(System.in);
6         bst bs = null;
7         int parm;
8
9         displayMenu();
10        int option = keyboard.nextInt();
11
12        while(option != 13){
13            if (option == 1){
14                int[] in = new int[20];
15                for(int i=0;i<20;i++){
16                    System.out.println("Enter an integer");
17                    in[i] = keyboard.nextInt();
18                }
19
20                bs = new bst(in);
21            }
22            else if (option == 2){System.out.println(bs.nodes());}
23            else if (option == 3){bs.leaves();}
24            else if (option == 4){System.out.println(bs.height());}
25            else if (option == 5){System.out.println(bs.odd());}
26            else if (option == 6){System.out.println(bs.zero());}
27            else if (option == 7){bs.descend();}
28            else if (option == 8){bs.ascend();}
29            else if (option == 9){System.out.println(bs.minimum());}
30            else if (option == 10){
31                System.out.println("Enter an integer");
32                parm = keyboard.nextInt();
33                bs.greater(parm);
34            }
35            else if (option == 11){
36                System.out.println("Enter an node's value");
37                parm = keyboard.nextInt();
38                System.out.println(bs.level(parm));
39            }
40            else if (option == 12){
41                System.out.println("Enter an node's value");
42                parm = keyboard.nextInt();
43                bs.path(parm);
44            }
45            else{System.out.println("Invalid Option");}
46
47            displayMenu();
48            option = keyboard.nextInt();
```

```
49     }
50
51
52 }
53 private static void displayMenu(){
54     System.out.println("1  - Enter a new BST\n" +
55         "2  - Count nodes\n" +
56         "3  - Print leaves\n" +
57         "4  - Print height\n" +
58         "5  - Sum odd values\n" +
59         "6  - Check for zero\n" +
60         "7  - Print in descending order\n" +
61         "8  - Print in ascending order\n" +
62         "9  - Print minimum value\n" +
63         "10 - Greater()\n" +
64         "11 - Find the level of a node\n" +
65         "12 - print the path to a node\n" +
66         "13 - Exit\n" +
67         "Choose an option - ");
68 }
69
70
71 private static String getParm(String parm){
72     Scanner keyboard = new Scanner(System.in);
73     String toReturn = "";
74     if(parm == "path"){
75         System.out.println("Enter the file path: ");
76         toReturn = keyboard.nextLine();
77     }
78     else if (parm == "stuData"){
79         System.out.println("Enter student id, name, age, and gpa sepearaed by space");
80         toReturn = keyboard.nextLine();
81     }
82     else if (parm == "id"){
83         System.out.println("Enter student ID #: ");
84         toReturn = String.valueOf(keyboard.nextInt());
85     }
86     else{
87         System.out.println("Weird Error, you better contact IT");
88     }
89     return toReturn;
90 }
91 }
92
```



```
1 public class bst
2 {
3     private class node{
4         int data;
5         node left;
6         node right;
7
8         private node(int data, node left, node right){
9             this.data = data;
10            this.left = left;
11            this.right = right;
12        }
13    }
14
15    node root;
16
17    public bst(int[] arr){
18        for(int i=0; i<20; i++){
19            insert(arr[i]);
20        }
21    }
22
23    public int nodes(){
24        return nodes(root);
25    }
26
27    public void leaves(){
28        leaves(root);
29    }
30
31    public int height(){
32        return height(root);
33    }
34
35    public int odd(){
36        return odd(root);
37    }
38
39    public boolean zero(){
40        return zero(root);
41    }
42
43    public void descend(){
44        descend(root);
45    }
46
47    public void ascend(){
48        ascend(root);
```

```
49     }
50
51     public int minimum(){
52         return minimum(root);
53     }
54
55     public void greater(int val){
56         greater(root, val);
57     }
58
59     public int level(int val){
60         return level(root, val, 1);
61     }
62
63     public void path(int val){
64         System.out.println(path(root, val, ""));
65     }
66
67     private boolean insert(int data){
68         if(root==null){//special case, insert at root
69             root = new node(data, null, null);
70         }
71         else{
72             node temp = root;
73             node prev = null;
74
75             while(temp != null){
76                 if(data == temp.data){
77                     return false;
78                 }
79                 else if(data > temp.data){
80                     prev = temp;
81                     temp = temp.right;
82                 }
83                 else{
84                     prev = temp;
85                     temp = temp.left;
86                 }
87             }
88             if(data > prev.data){
89                 prev.right = new node(data, null, null);
90             }
91             else{
92                 prev.left = new node(data, null, null);
93             }
94         }
95         return true;
96     }
```

```
97
98     private int nodes(node temp){
99         if(temp == null){
100             return 0;
101         }
102         else{
103             return 1 + nodes(temp.left) + nodes(temp.right);
104         }
105     }
106
107     private void leaves(node temp){
108         if(temp == null){
109             return;
110         }
111         else if(temp.left == null && temp.right == null){
112             System.out.println(temp.data);
113         }
114         else{
115             leaves(temp.left);
116             leaves(temp.right);
117         }
118     }
119
120     private int height(node temp){
121         if(temp == null){
122             return 0;
123         }
124         int lheight = height(temp.left);
125         int rheight = height(temp.right);
126         if(lheight > rheight){
127             return 1 + lheight;
128         }
129         else{
130             return 1 + rheight;
131         }
132     }
133
134     private int odd(node temp){
135         if(temp == null){
136             return 0;
137         }
138         else if(temp.data % 2 == 1){
139             return temp.data + odd(temp.left) + odd(temp.right);
140         }
141         else{
142             return odd(temp.left) + odd(temp.right);
143         }
144     }
```

```
145
146     private boolean zero(node temp){
147         if(temp == null){
148             return false;
149         }
150         else if(temp.data == 0){
151             return true;
152         }
153         else{
154             return (zero(temp.left) || zero(temp.right));
155         }
156     }
157
158     private void descend(node temp){
159         if(temp == null){
160             return;
161         }
162         else{
163             descend(temp.right);
164             System.out.println(temp.data);
165             descend(temp.left);
166         }
167     }
168
169     private void ascend(node temp){
170         if(temp == null){
171             return;
172         }
173         else{
174             descend(temp.left);
175             System.out.println(temp.data);
176             descend(temp.right);
177         }
178     }
179
180     private int minimum(node temp){
181         if(temp == null){
182             return Integer.MAX_VALUE;
183         }
184         else{
185             int toReturn;
186             int lmin = minimum(temp.left);
187             int rmin = minimum(temp.right);
188             if(lmin < rmin && lmin < temp.data){
189                 toReturn = lmin;
190             }
191             else if(rmin < lmin && rmin < temp.data){
192                 toReturn = rmin;
```

```
193     }
194     else{
195         toReturn = temp.data;
196     }
197     return toReturn;
198 }
199 }
200
201 private void greater(node temp, int val){
202     if(temp == null){
203         return;
204     }
205     else if(temp.data > val){
206         System.out.println(temp.data);
207     }
208     greater(temp.left, val);
209     greater(temp.right, val);
210 }
211
212 private int level(node temp, int val, int lvl){
213     if(temp == null){ //didn't find it
214         return -1;
215     }
216     else if(temp.data == val){ //found it
217         return lvl;
218     }
219     else{ //keep searching
220         if(val > temp.data){ //go right
221             return level(temp.right, val, lvl+1);
222         }
223         else{ //go left
224             return level(temp.left, val, lvl+1);
225         }
226     }
227 }
228
229 private String path(node temp, int val, String path){
230     if(temp == null){ //no path existed
231         return "Path did not exist";
232     }
233     else if(temp.data == val){ // found it
234         return path + temp.data;
235     }
236     else{ // keep searching
237         if(val > temp.data){ //go right
238             return path(temp.right, val, path+temp.data+"-->");
239         }
240         else{ //go left
```

```
241         return path(temp.left, val, path+temp.data+"-->");
242     }
243 }
244 }
245 }
246
```