CSE143X Section #16 Problems

For the following problems, assume that you are using a binary tree of ints defined as follows:

```
        public class IntTreeNode {
            public int data;          // data stored at this node
            public IntTreeNode left;  // reference to left subtree
            public IntTreeNode right; // reference to right subtree

            // post: constructs a leaf node with given data
            public IntTreeNode(int data) {
                this(data, null, null);
            }

            // post: constructs a branch node with the given data and links
            public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {
                this.data = data;
                this.left = left;
                this.right = right;
            }
        }

        public class IntTree {
            private IntTreeNode overallRoot;

          <methods>
        }
```

For each problem below, you are to write a new public method for the IntTree class that performs the given operation. You may define additional private methods to implement your public methods.

1. Write a method numNodes that returns the total number of nodes in the tree.

2. Write a method numLeaves that returns the total number of leaf nodes in the tree.

3. Write a method equals that compares two binary trees of integers to see if they are equal to each other (i.e., if they have the same structure and store the same values). For example, if variables of type IntTree called t1 and t2 have been initialized, then t1.equals(t2) will return true if the trees are equal and will return false otherwise. Two trees are considered equal if each node in either tree has a corresponding node in the other tree in the same location relative to the root and storing the same value. If both trees are empty, they are considered equal.

4. Write a method depthSum that returns the sum of the values stored in a binary tree of integers weighted by the depth of each value. Return the value at the root plus 2 times the values stored at the next level of the tree plus 3 times the values stored at the next level of the tree plus 4 times the values stored at the next level of the tree and so on.

   For example, in the tree below:

```
                    +---+
                    | 9 |
                    +---+
                   /     \
                  /       \
              +---+       +---+
              | 7 |       | 6 |
              +---+       +---+
             /     \           \
            /       \           \
        +---+       +---+      +---+
        | 3 |       | 2 |      | 4 |
        +---+       +---+      +---+
                    /               \
                   /                 \
               +---+                 +---+
               | 5 |                 | 2 |
               +---+                 +---+
```

   The sum would be computed as:

```
        1 * 9 + 2 * (7 + 6) + 3 * (3 + 2 + 4) + 4 * (5 + 2) = 90
```

5. Write a method height that returns the height of a binary tree. The height of a tree is defined to be the number of levels it has (i.e., the number of nodes along the longest path from the root to a leaf). The empty tree has a height of 0. A tree of one node has a height of 1. A root node with one or two leaves as children has a height of 2. And so on.

6. Write a method isFull that returns whether or not a binary tree is full (true if it is, false otherwise). A full binary tree is one in which every node has 0 or 2 children. Below are examples of each.

```
                  full tree                        not a full tree
                    +---+                              +---+
                    | 2 |                              | 7 |
                    +---+                              +---+
                   /     \                            /     \
                  /       \                          /       \
              +---+       +---+                  +---+       +---+
              | 3 |       | 1 |                  | 4 |       | 0 |
              +---+       +---+                  +---+       +---+
             /     \                            /     \           \
            /       \                          /       \           \
        +---+       +---+                  +---+       +---+      +---+
        | 8 |       | 7 |                  | 9 |       | 2 |      | 8 |
        +---+       +---+                  +---+       +---+      +---+
```

    By definition, the empty tree is considered full.

7. Write a method contains that takes an integer n as a parameter and that returns true if the tree contains the value and that returns false otherwise.

8. Write a method printLevel that takes an integer n as a parameter and that prints the values at level n from left to right. The values should be printed to System.out, one per line. We will use the convention that the overall root is at level 1, that it's children are at level 2, and so on. For example, if a variable t stores a reference to the following tree:

```
                       +----+
                       | 12 |
                       +----+
                     /        \
                   /            \
              +----+            +----+
              | 19 |            | 93 |
              +----+            +----+
             /      \                 \
            /        \                 \
        +----+      +----+            +----+
        | 11 |      | 14 |            | 15 |
        +----+      +----+            +----+
                   /
                  /
              +----+
              | 10 |
              +----+
```

then the call:

```
     t.printLevel(3);
```

would produce the output:

```
     11
     14
     15
```

If there are no values at the level, your method should produce no output. Your method should throw an IllegalArgumentException if passed a value for level that is less than 1.

9. Write a method called hasPathSum that takes an integer n as a parameter and that returns true if there is some path from the overall root of a tree to a node of the tree in which the sum of the data stored in the nodes adds up to n (returning false if no such path exists). For example if the variable t refers to the following tree:

```
                              +----+
                              |  5 |
                              +----+
                            /        \
                          /            \
                   +----+                +----+
                   |  1 |                | 21 |
                   +----+                +----+
                  /      \                     \
                 /        \                     \
             +----+       +----+               +----+
             | -9 |       |  2 |               | 20 |
             +----+       +----+               +----+
            /      \                          /      \
           /        \                        /        \
       +----+      +----+                +----+      +----+
       |  3 |      | 30 |                | 13 |      |  4 |
       +----+      +----+                +----+      +----+
```

Below are various calls and an explanation for the value returned:

```
     t.hasPathSum(8) returns true because of the path (5, 1, 2)
     t.hasPathSum(26) returns true because of the path (5, 21)
     t.hasPathSum(0) returns true because of the path (5, 1, -9, 3)
     t.hasPathSum(5) returns true because of the path (5)
     t.hasPathSum(1) returns false because no path with that sum exists
```

10. Write a method writeTree that writes the tree to System.out in a specific format. You are to perform a preorder traversal of the tree, producing exactly one line of output for each node. Each output line should begin with a code to indicate what kind of node it is followed by the data stored in the node. The different kinds of nodes are indicated below:

Code	Node Kind
0	leaf node (no children)
1	branch node with left child only (empty right)
2	branch node with right child only (empty left)
3	branch node with nonempty left and right children

For example, below is a binary tree and the output it would produce:

```
        Sample Tree                     Output Produced
            +---+                               3 7
            | 7 |                               1 9
            +---+                               0 5
           /     \                              3 8
          /       \                             2 4
      +---+       +---+                         0 9
      | 9 |       | 8 |                         0 6
      +---+       +---+
     /           /     \
    /           /       \
+---+        +---+     +---+
| 5 |        | 4 |     | 6 |
+---+        +---+     +---+
                  \
                   \
                  +---+
                  | 9 |
                  +---+
```