Note: how to build an AVL tree

1. Insert():
use AvlNode<AnyTpye> insert(val) instead of return void, because if root = null, need to return new root. For the same reason, when recursively call insert(), don't forget: t.left = insert(val, t.left).

if(t == null) return new AvlNode<>(x, null, null); to create new node.

don't forget to return balance(t) instead of t at last, reason see Balance()

2. Remove():
As well AvlNode<AnyTpye> remove(val) is necessary, because the root could be removed, need to return new root. For the same reason, when search node, don't forget t.left = remove(val, t.left).

When find the node, there are two condition: (1) both t.left & t.right != null (2) at least one of them is null.

don't forget to return balance(t) instead of t at last, reason see Balance()

3. Balance():
if (right side higher), use height(t.right.right) >= height(t.right.left) will NOT cause OUTBOUNDEXCEPTION, because right side higher, t.right != null.

if not balanced, call t = rotate/doubleWithLeft/RightChild(t) to update t.

don't forget to add: t.height = Math.max(height(t.left), height(t.right)) + 1;

How do we know every node's height? Because every time call insert(t), insert(t) will call insert(t.left/right), and insert(t.left/right) will call balance(t.left/right), and balance will set the height from the bottom to top, update all the ancient nodes.

null node's height = -1

4. Contains():
simple one, use a while(t != null) to check.

5. printTree():
simple one, if (t != null){ use any of pre/in/post order}.

6. rotateWithLeftChild():
when it is called, it is like: t = rotate/doubleWithLeft/RightChild(t). So don't have to use the pre.t to point t, because, just return new t is fine.

don’t forget to update two nodes: (other node below t and t.left will not be changed, node above will be updated in outer balance())

```
t.height = Math.max( height( k1.left ), height( k1.right ) ) + 1;
t.left.height = Math.max( height( k2.right ), k1.height ) + 1;
```

7. doubleWithLeftChild():
A little tricky that call rotateWithLeftChild() twice, remember rotate left child first, then parent node. It is like:

```
t.left = rotateWithRightChild( t.left );
return rotateWithLeftChild( t );
```