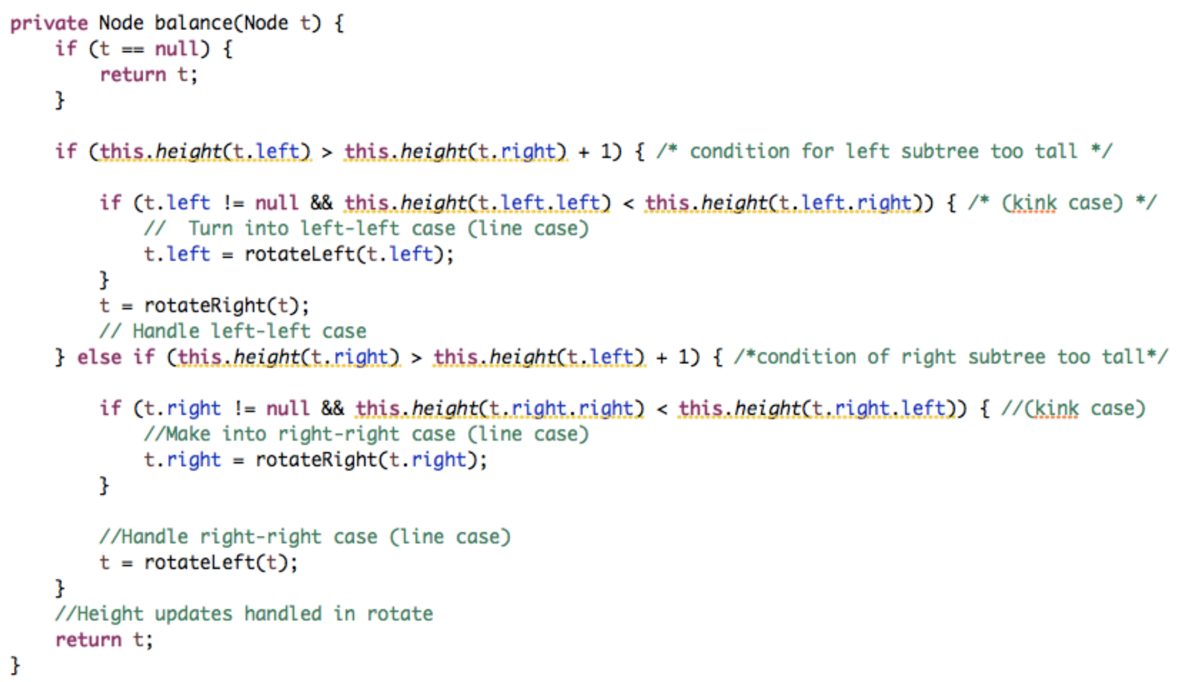
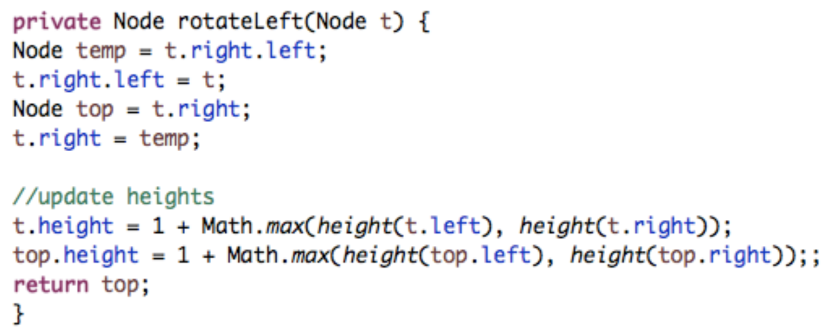
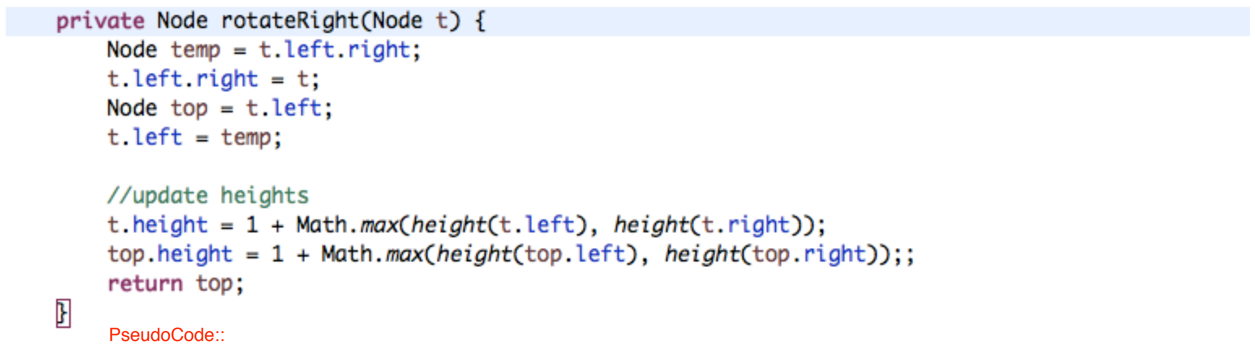


```java
private Node balance(Node t) {
    if (t == null) {
        return t;
    }

    if (this.height(t.left) > this.height(t.right) + 1) { /* condition for left subtree too tall */

        if (t.left != null && this.height(t.left.left) < this.height(t.left.right)) { /* (kink case) */
            //  Turn into left-left case (line case)
            t.left = rotateLeft(t.left);
        }
        t = rotateRight(t);
        // Handle left-left case
    } else if (this.height(t.right) > this.height(t.left) + 1) { /*condition of right subtree too tall*/

        if (t.right != null && this.height(t.right.right) < this.height(t.right.left)) { //(kink case)
            //Make into right-right case (line case)
            t.right = rotateRight(t.right);
        }

        //Handle right-right case (line case)
        t = rotateLeft(t);
    }
    //Height updates handled in rotate
    return t;
}
```



```java
private Node rotateLeft(Node t) {
Node temp = t.right.left;
t.right.left = t;
Node top = t.right;
t.right = temp;

//update heights
t.height = 1 + Math.max(height(t.left), height(t.right));
top.height = 1 + Math.max(height(top.left), height(top.right));;
return top;
}
```



```java
private Node rotateRight(Node t) {
    Node temp = t.left.right;
    t.left.right = t;
    Node top = t.left;
    t.left = temp;

    //update heights
    t.height = 1 + Math.max(height(t.left), height(t.right));
    top.height = 1 + Math.max(height(top.left), height(top.right));;
    return top;
}
```

PseudoCode::

```
RotateLeft (Node t)

give right child's left subtree to parent,
and put parent below right child (move right child up)
adjust heights of parent/right child so they are accurate

rotateRight(Node t)

give left child's right subtree to parent
put parent below left child (move left child up)
adjust heights of parent/ old left child so they are accurate again

balance(Node t)

if t = null: do nothing
if left subtree is more than 1 higher than right:
    if there is a kink
        rotate left child left
    rotate node right
else if right subtree is more than 1 higher than left
    if there is king
        rotate right child right
    rotate node left
return updated node/subtrees (heigh changes handled in rotateLeft/Right
```