From Rotations to Red-Black Trees

Assuming that you've completed the code to rotate subTrees, let's look at how we might extend our Binary Tree to become a fully-fledged Red-Black Tree.

Here's my updated `insert()` method:

```
/**
 * This is the public insert method, i.e. the one that the outside world will invoke.
 * It then kicks off a recursive method to "walk" down through the tree - this is
 * possible because each sub-tree is itself a tree.
 * @param value Object to insert into the tree
 */
public void insert(T value){
        Node<T> node = new Node<T>(value); // Create the Node to add
```

NB: Your code will not require <T> for variables (you should delete to compile)

```
        //Special case that cannot be handled recursively
        if ( root == null ) {
                root = node;

                //Remember that new nodes default to Red but
                //the root must always be black
                node.nodeColourRed = false;
                return;
        }

        //Initially we start at the root. Each subsequent recursive call will be to a
        //left or right subtree.
        insertRec(root, node);

        //Now that we've inserted we need to make it Red-Black (if necessary)
        handleRedBlack(node);

}
```

I'll give you the first little bit of code, since it will terminate any recursion that might be invoked:

```
/**
 * Note: This method may be called recursively but only for the case
 * where the Uncle is red (assuming that the parent node is red - which
 * is a violation, of course)
 * @param newNode
 */
void handleRedBlack(Node<T> newNode)
{
        //terminating case for "back" recursion - e.g. case 3 (video)
        if(newNode == root)
        {
                newNode.nodeColourRed = false;
                return;
        }

        Node<T> uncle;
        Node<T> parent = newNode.parent;
        Node<T> grandParent = parent.parent;
        //Now that it's inserted we try to ensure that it's a RedBlack Tree
        //Check if parent is red. This is a violation. I (the new node) am red
        //so my parent cannot also be red!
```

Also, you can see that I've declared variables that will allow me to easily map to the violation example case. `parent` cannot be null. `grandParent` could be null, which actually is okay. However, I don't initialise `uncle` yet because that could easily cause an exception (plus, I don't know if the uncle will be on the left or right side of grandParent yet.)   Note: I haven't shown you how to initialise the parent field. Hint: you do it when you're perform the actual insertion of the node.

The only way that the new node can be a violation is if its parent is red (after that, it's a case of figuring out which action to take to fix the problem; the two possibilities are that it **has a Red Uncle** (which doesn't involve any rotations to fix but there's a recursive element to it) OR that it **has a Black Uncle** (which has four possible sub-scenarios). Once again, I'll refer you to http://www.geeksforgeeks.org/red-black-tree-set-2-insert/ for explanations/illustrations of these scenarios and subsequent actions.

I'll give you the first bit to get you started and then leave you to it!

```
//Now that it's inserted we try to ensure that it's a RedBlack Tree
//Check if parent is red. This is a violation. I (the new node) am red
//so my parent cannot also be red!
if(parent.nodeColourRed)
{
        //important that we figure out where the uncle is
        //relative to the current node
        if(uncleOnRightTree(newNode))
        {
                uncle = getRightUncle(newNode);
        }
        else
        {
                uncle = getLeftUncle(newNode);
        }

        //Now we need to check if x's uncle is RED (Grandparent must
        //have been black)
        //This is case 3 according to the video
        //(https://www.youtube.com/watch?v=g9SaX0yeneU)
        if((uncle != null) && (uncle.nodeColourRed))
        {
                //this case is not too bad.
                //it involves recolouring and then recursing

                //CODE OMITTED  - it's only 4 lines!
        }
```