AADS Lab week 5

Integrate Swing Viewer

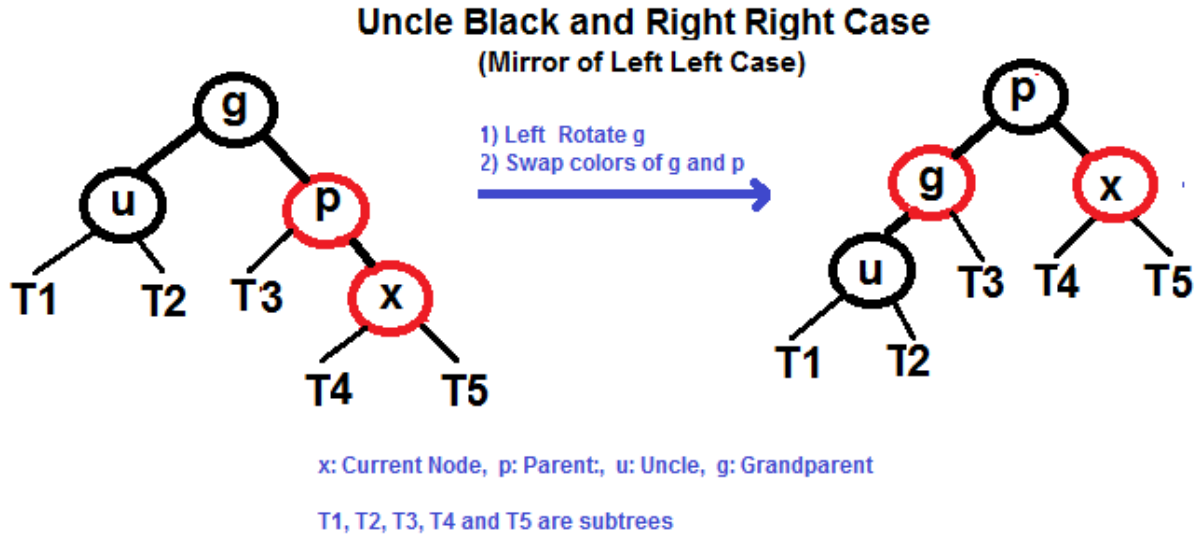
Hopefully, this should be relatively straightforward. The main changes should be changing the package name and making some of the instance fields in the RedBlackTree `protected`.

Enhancements:

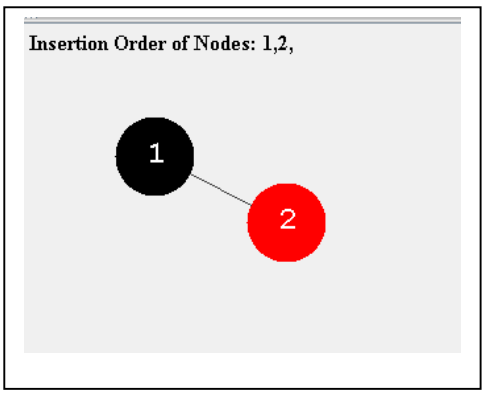
- I forgot to clear the string that maintains the insertion order when the Clear Button is pressed.
- Clear the JTextField after each insert.

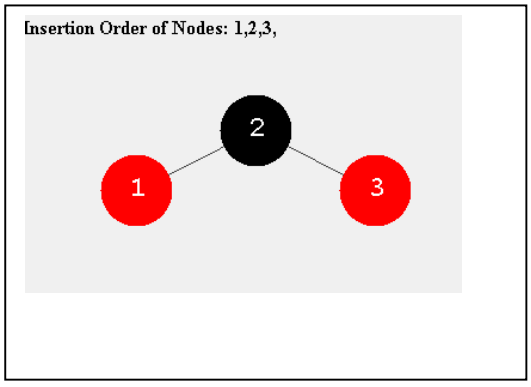
Of course, you should be able to test and verify the situations that you've implemented. At this stage your tree should be able to handle a Red Uncle. Last week you started work on either the Right-Right or Left-Left cases for a black uncle.

Right-Right Case continued



Last week you may even have constructed some code which handles a Right-Right case for a basic tree which creates a R-R scenario, e.g. Insert(1,2,3).



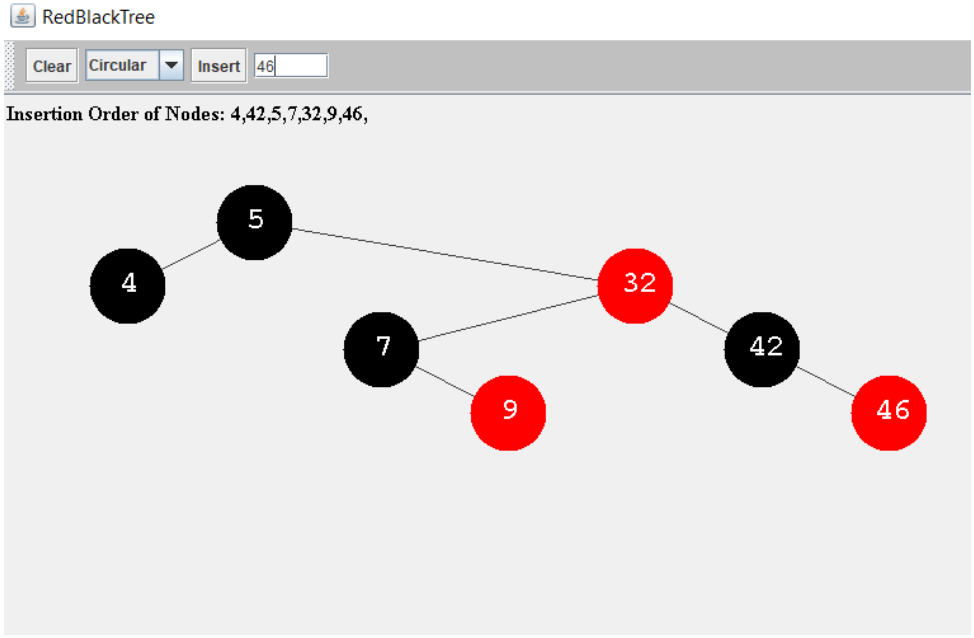


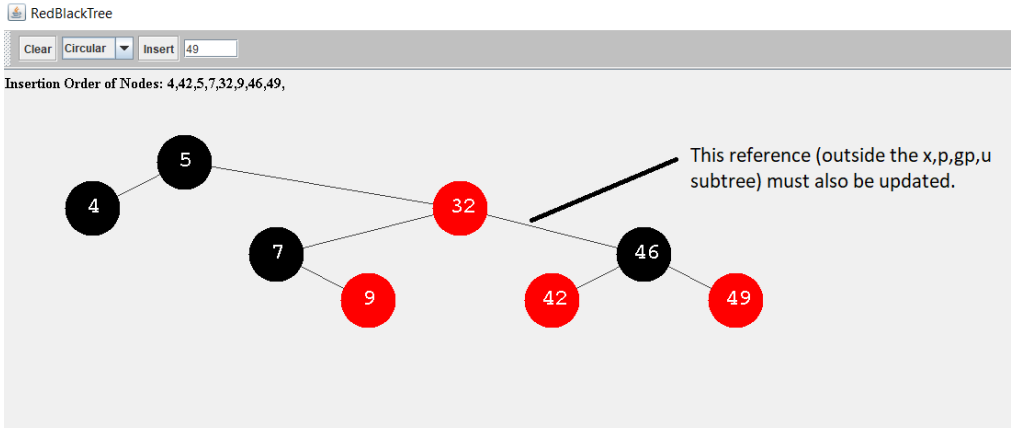
I asked you to write it as a method call and left the parameter list and return value for you to think about.
Perhaps you thought, "I'll pass in the parent and grandparent references since they are involved in the rotation and both their colours need to be swapped".

```
private void applyRightRightCase(Node inParent, Node inGrandParent)
```

And this could work for our test case scenario above…. **assuming that your code remembered to update the root accordingly**.

However, even if you got the code to work with the method signature shown, it's not scalable. To illustrate, I'll show another two screen-shots of a before/after Red-Red situation.





As you can see, the Red-Red situation is always handled with respect to the parent, grandparent and uncle subtree, **but this always has an impact on the parent of the entire subtree because it has a new subtree root**.

So, we know that we must update the parent of the subtree (which is effectively a great-grandparent is we continue the analogy).

We can adapt our first attempt at our method call in two ways:

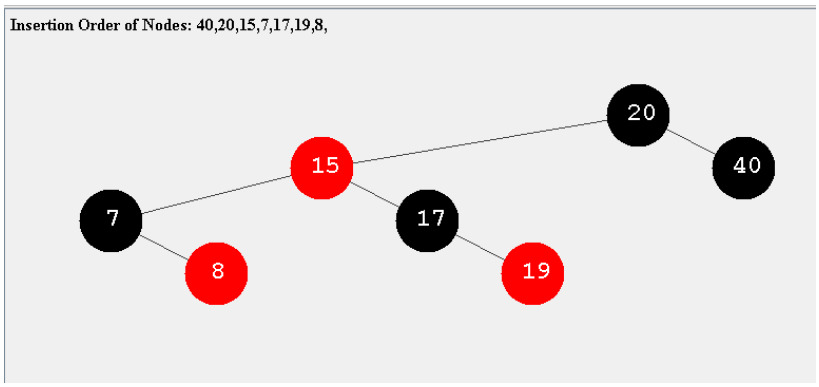
1. `void applyRightRightCase(Node inParent, Node inGrandParent Node inSubtreeParent).`
   We can pass in a reference to that node and update it internally within the method.
2. `Node applyRightRightCase(Node inParent, Node inGrandParent)`
   We can return the new subtree root to the calling code. It will then update the subtree parent reference using this information.

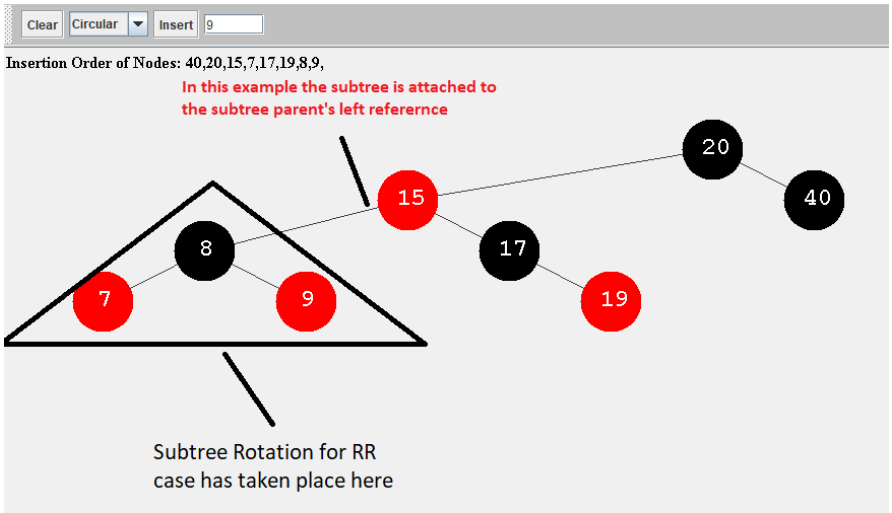
You can use either of the two mechanisms, depending on which one appeals more to your logic.

Updating the subtree parent

There are three scenarios:

1. The subtree is the tree (just as in the first pair of screenshots). The root is updated.
2. The subtree is attached to the right of the subtree's parent. See the second pair of screenshots.
3. The subtree is attached to the left of the subtree's parent. See the pair of screenshots below.

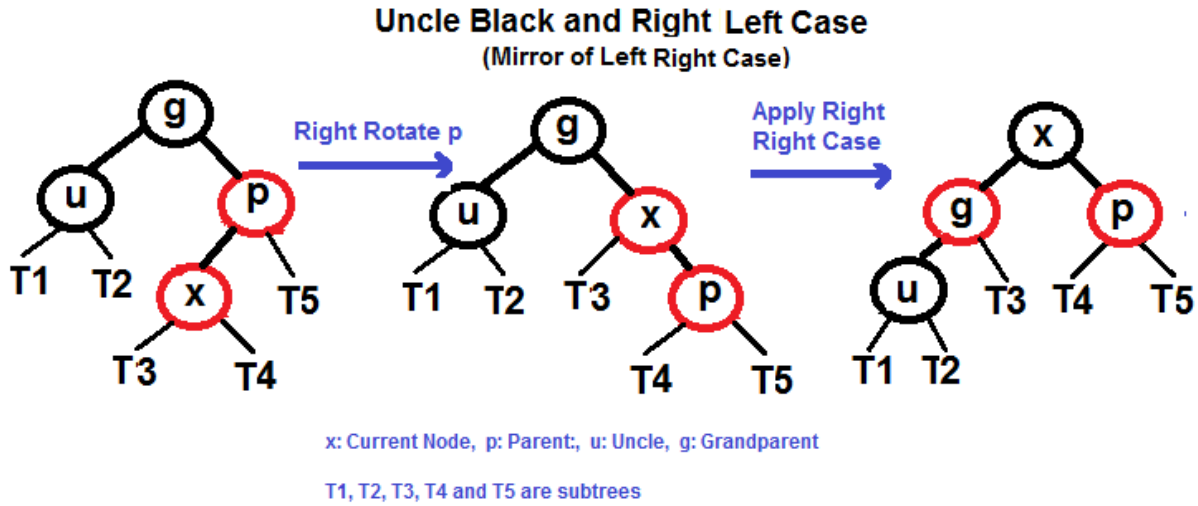




Other black uncle cases

Once you've implemented Right-Right, Left-Left is trivial (see the appendix at the end of the week 4 document for the various cases).
Right-Left and Left-Right are mirror images of each other, so once you've solved one it's again trivial to implement the other.

If you look at the Right-Left case, for example:



All we have to do is:

1. Call our method to do a rightRotation around the parent (assign the return value to the grandParent's right reference.)
2. Apply the Right-Right case to the resultant tree (and we've just looked at that in some detail).

Caveat: Since the relative positions of x and p will change from the first stage of the process, we need to be careful!