Recursion in Binary Trees - Use Debugger as an aid to understanding

In terms of understanding how recursion works, there are two techniques that I would recommend:

1. Hand trace through an algorithm. It's usually a pen-and-paper exercise. Go through a code listing for a scenario (e.g. inserting an element into some pre-existing binary tree). Draw, write, annotate... whatever you feel helps *you* to step though and follow the logic.
2. Use the debugger (on code that already know works - you're not debugging, you're trying to enhance your understanding.) I'll elaborate below.

I want to try and understand how a recursive `findMinimum()` algorithm works.

One thing that newcomers to recursion can find difficult to conceptualise is the actual calling mechanism and the fact that no instance of the method can finish until the one that they've called returns back to them.

This can only happen when one of them eventually hits the "**end condition**". As you should be aware, every recursive method must have some such condition. Otherwise you get infinite recursion which always manifests itself in the stack-memory overflowing.

In the first screenshot I've set a breakpoint on the end-condition because it will allow me to see the state of everything at that point.

In the Debug panel you can see stack-frame: `main()` called `findMinimum()` which called `recFindMinimum()` which resulted in a number of recursive calls.

You can select any of these calls to see the state of subTreeRoot (the recursive parameter) at that point. In the screenshot, I'm looking at the last call. I can clearly see that the node has no leaf nodes (left and right are null) and the value is 10 (I'm running the example from the lab, which I've also included in Appendix A). This - the value 10 - is obviously what the green highlighted code is about to return (see line 172).

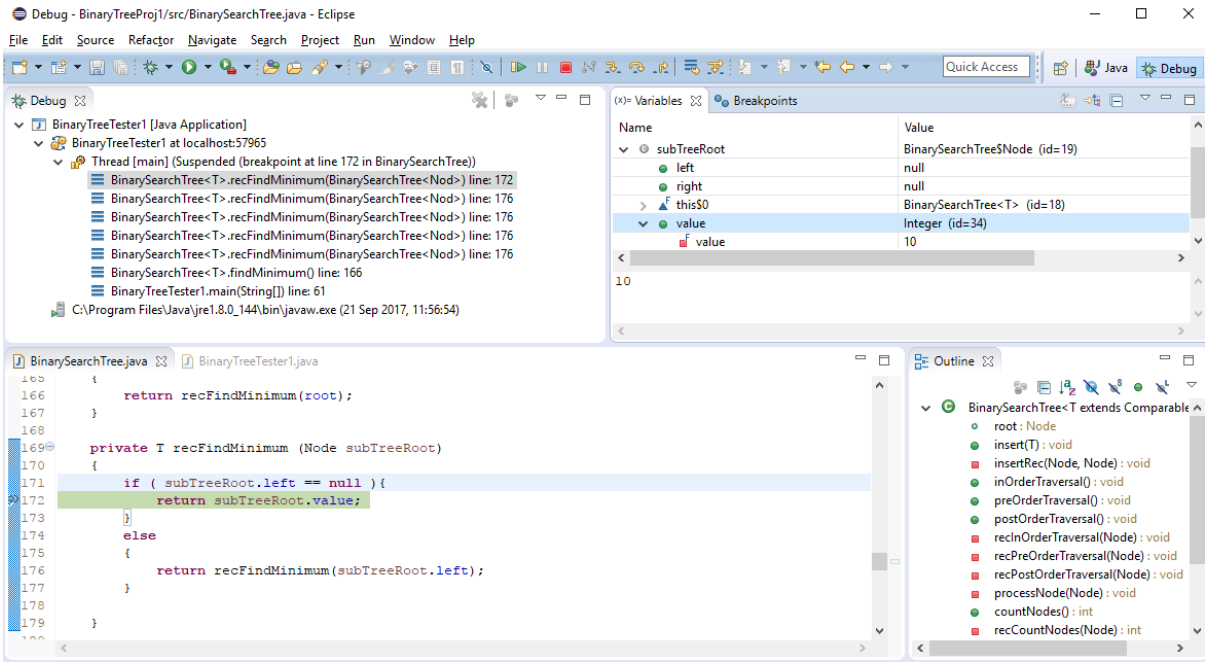

Figure 1: recursion has hit the end-condition and is about to "unwind" itself

Now let's execute that instruction (use Step-Over button or F6) to see what changes...

Figure 2 shows the next state (if you can compare the two side-by-side it may be useful).

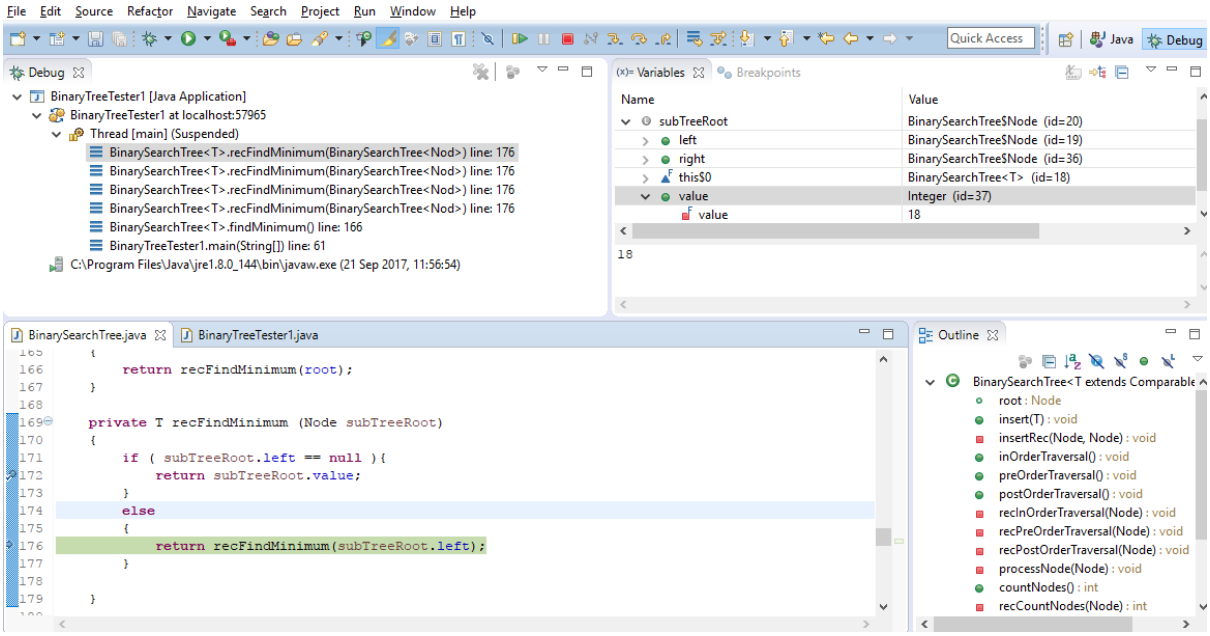

Figure 2: "unwinding" about to begin in earnest

The stack frame is now one level smaller because the deepest recursive call has **returned**.

Returned to where?

To the line that called it - the green highlighted line (that's actually when we were at the node with value 18, as you can see.)

And what is it doing?

It's simply returning what it got from the recursive call that has just finished, i.e. the value 10.

And that's exactly what each of the other recursive calls are also doing (they're all waiting patiently at line 176 to finish up!)

Exercise:

If you want to see the stack-frame building up - i.e. as it traverses the tree to find the minimum - then set an additional breakpoint on line 176 and restart the program under Debugger control. Note that each time it ***breaks***, you can use the Resume button (or F8) to continue when you're ready to.

Appendix A: A partial listing of the test code that was used

```
BinarySearchTree<Integer> myTree = new BinarySearchTree<Integer>();

myTree.insert(40);
myTree.insert(32);
myTree.insert(37);
myTree.insert(34);
myTree.insert(26);
myTree.insert(29);
myTree.insert(18);
myTree.insert(20);
myTree.insert(10);
myTree.insert(49);
myTree.insert(60);
myTree.insert(70);
myTree.insert(80);
myTree.insert(75);
myTree.insert(55);


System.out.println("Find Minimum: "+myTree.findMinimum());
```