1. *lowest* in line 43 of AbstractBalancedBinarySearchTree equals
    a. If Node A has only one Child
        i. *removeSelf(N node)* replaces Node A with its Child
        ii. *returns* the Parent of Node A that was removed
    b. If Node A has both Childs
        i. *removeHibbard(N node)* replaces Node A with the node with the lowest key in the subtree of Node A's right Child, then reconnects the left Child to that node and assigns its parent as its new right child
        ii. *returns* the lowest node which has an updated subtree, either the node that replaces Node A or that node's right child
2. *balance* method
    a. AVL Tree – ensures balance after removal by measuring balance factor and then adjusting tree left or right accordingly
        i. Checks balance factor (BF) of Node A (*lowest*)
        ii. If BF = 2, examine left Child of Node A
            1. If BF of left Child is == -1, the left Child B has only a right child
                a. *rotateLeft* replaces the left Child B with its right Child C, and then sets the left child of C to B
            2. Otherwise the left Child has only a left child
                a. *rotateRight* replaces the right Child B with its left Child C, and then sets the right child of C to B
        iii. If BF = -2, examine right Child of Node A
            1. If BF of right Child is == -1, the right Child B has only a left child
                a. *rotateRight* replaces the right Child B with its left Child C, and then sets the right child of C to B
            2. Otherwise the left Child has only a right child
                a. *rotateLeft* replaces the left Child B with its right Child C, and then sets the left child of C to B
        iv. Recursively checks this with Parent of Node A until at root
    b. Red-Black Tree
        i. Similar process to AVL Tree, but checks Node A's Parent and Uncle to see if they are black or red and then rotates Node A's Parent and Grandparent accordingly to ensure following specifications:
            1. The root and all leaves (null) are black.
            2. Every red node must have two black child nodes.
            3. Every path from a given node to any of is descendant leaves must contain the same number of black node
        ii. Recursively checks this with Parent of Node A until at root

Generally, the remove method in AbstractBalancedBinarySearchTree removes the correct node, reconnects it to maintain BST integrity, then uses a set of rules (either height / balance factor or colors that enforce symmetry) to rebalance the tree. This ensures proper removal that maintains a BST with O(log N) search.