AVL Trees

AVL Tree is the perfectly balanced Binary Search Tree (BST). AVL is named after its two Soviet inventors, Georgy Adelson-Velsky & Evgenii Landis. If we have a BST and the nodes are in ascending / desending order the time complexity of searching / deletion will be big O of h O(h) and it is imbalance from left or right [left-heavy / Right-heavy], to make this more optimize we check balance factor [height of left sub tree - height of right sub tree] and if it is equal to [-1, 0, 1] the tree is balanced if BF does not lies in that range Tree is imbalance to make tree balance we perform rotations.

→ Operations:

(i) Insertion: The insertion in AVL Trees is same as BST compare data with root if greater insert in right if less insert in left. and after every insertion check BF if the node became imbalance perform rotation on critical node.

→ The duplicates are not allowed in AVL Trees.

Deletion: The deletion in AVL Trees is also same as BST there are 3 cases:

Case 1:

Deleting leaf node: To delete a leaf node we make it null and check BF if it is imbalance perform rotation.

Case 2:

Possibility - I: Deleting node have only left child, we link lef child to the parent of deleting node.

Possibility - II: Deleting node have right child, we link right child to parent of deleting node.

Case 3: The deleting node has both children then we find the inorder predecessor or inorder successor of the deleting node and Link it with parent of ddeting node.

In Last we check BF for performing rotation.

Rotation:

The rotation in AVL Trees in performed when any nodes of tree become imbalance which is checked by balance factor of every node there are four rotations and rotations are performed on three nodes only:

1. **LL (Left to Left) Rotation:** The LL Rotation is performed when any node is inserted in (Left sub tree's Left) or deleted that affect the height of the tree.

2. **RR (Right to Right) Rotation:** The RR Rotation is performed when any node is inserted in (right subtree's right) or deleted that affect the height of the tree.

For LL and RR Rotation: We make a single rotation in LL and RR Rotation. In LL Rotation we rotate the nodes to the right and in RR Rotation we rotate to left. In LL Rotation middle node will be the root and upper node will be the right sub tree and lower will be left sub tree and in RR Rotation middle node will be the root and upper

node will be the left sub tree and lower will be right sub tree.

3- LR (Left to Right) Rotation:

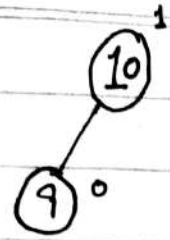
The LR Rotation is performed when any node is inserted in (right of left sub tree) or deleted that affect the height of the tree.

4- RL (Right to left) Rotation):

The RL Rotation is performed when any node is inserted in (left of right sub tree) or deleted that affect the height of the tree.

For LR and RL Rotation:

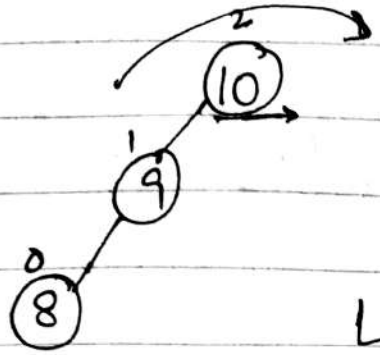
We make double rotations for LR we perform right rotation first then left rotation and in RL rotation we perform left rotation first than right rotation.



→ Binary Search Tree / AVL Tree

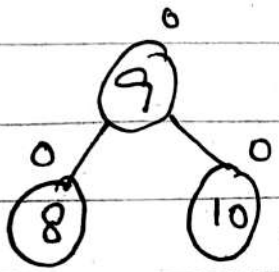
Balance Factor 0, 1.

(8) → inserting.

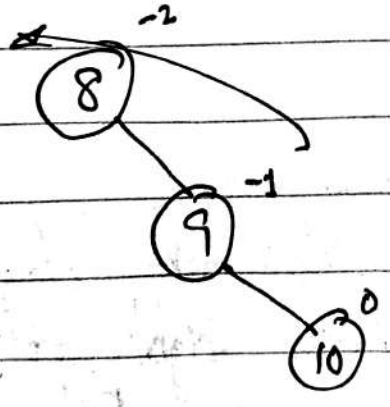


10 is critical node cause BF is 2 we inserted in left of left Sub-Tree

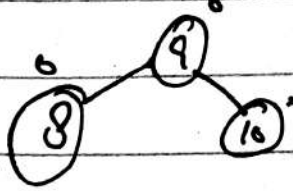
LL Rotation.



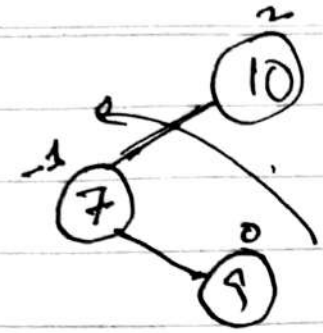
Tree Balanced.



RR Rotation.

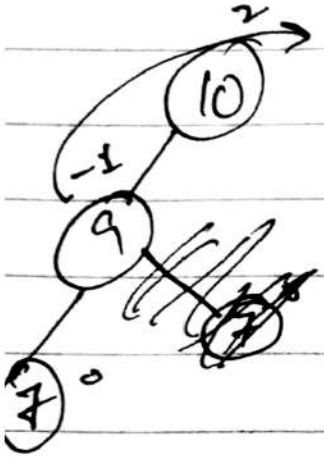


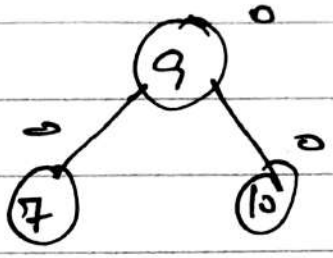
Balanced

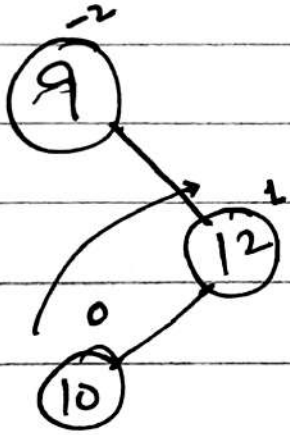


Inserted in Right of left Sub Tree.
Perform LR Rotation.

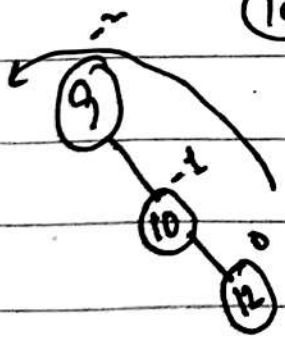
→ Double Rotation

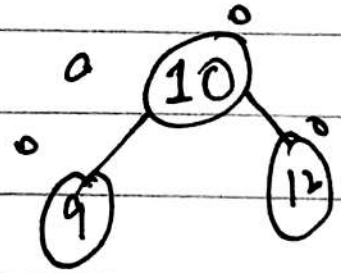




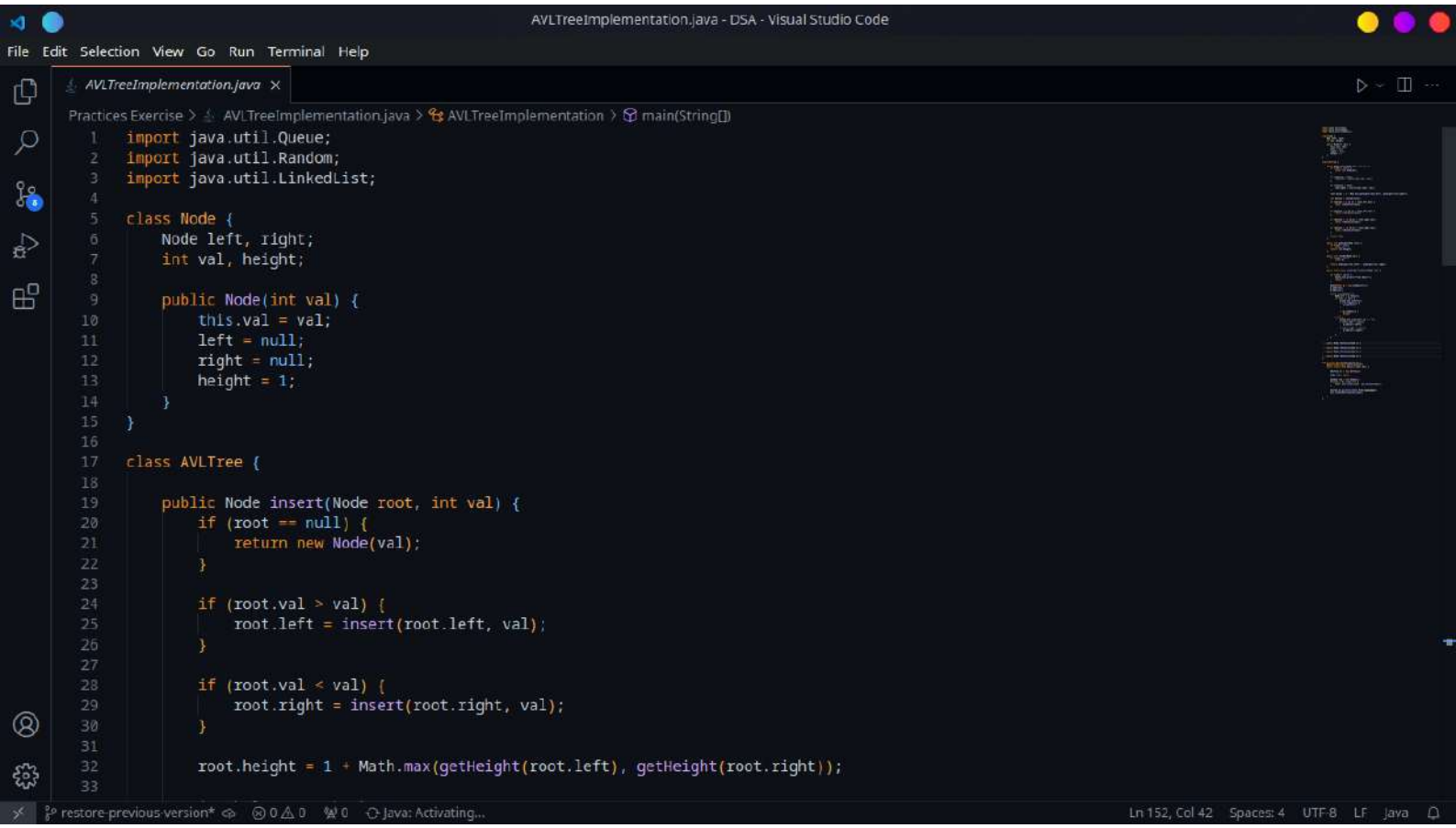


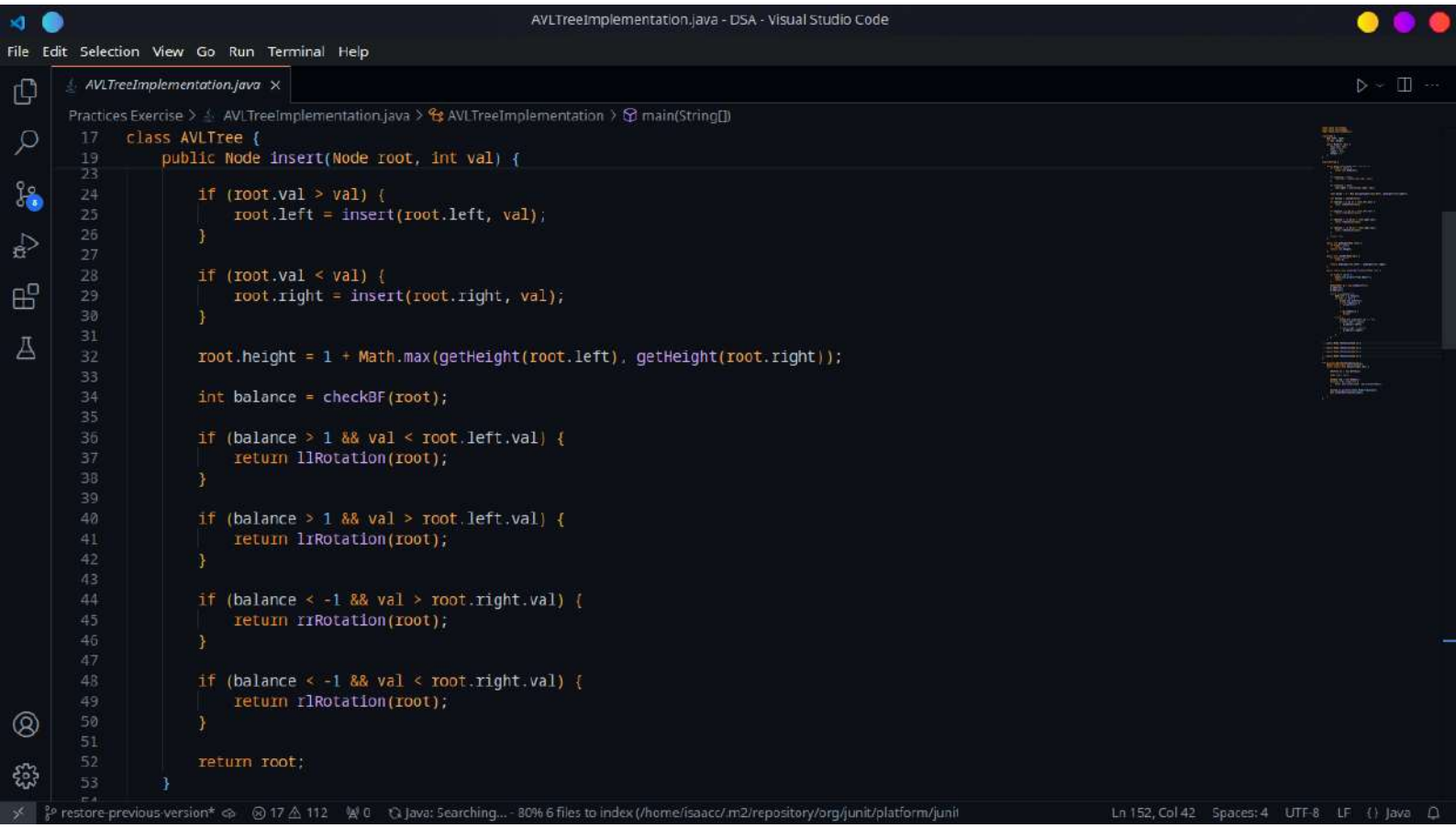
Inserted in Right's left.
Perform RL Rotation
→ Double Rotation

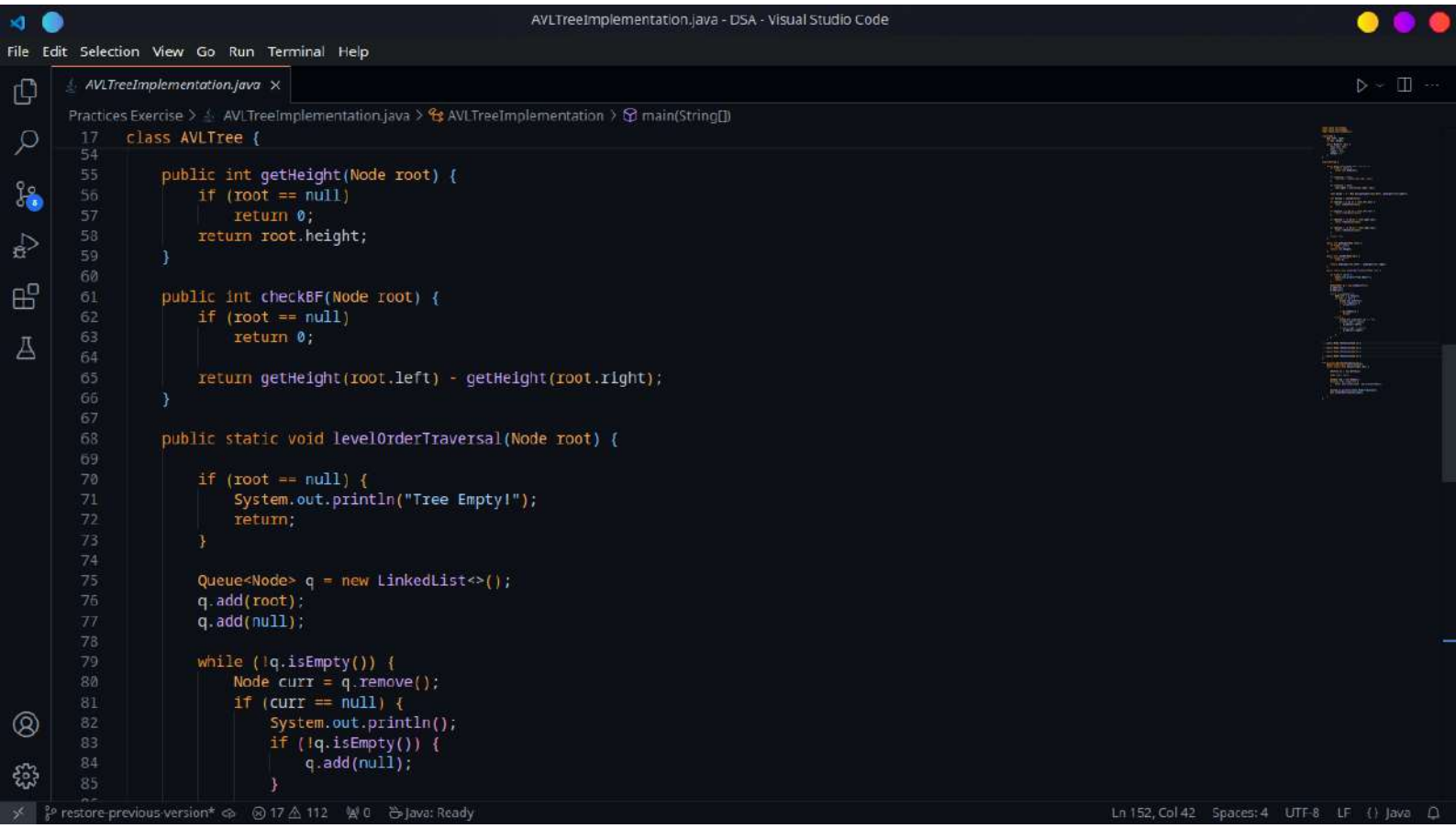


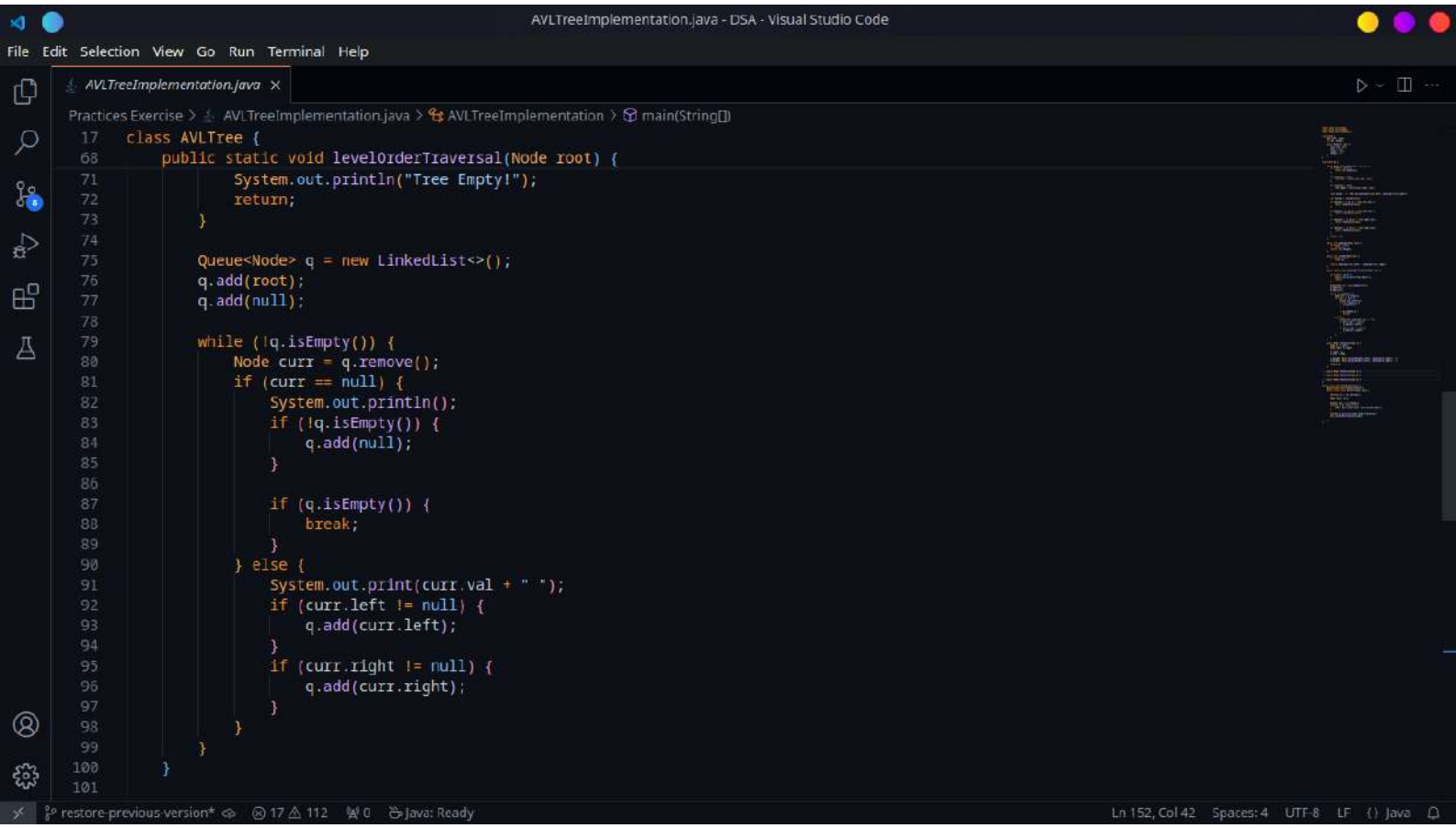


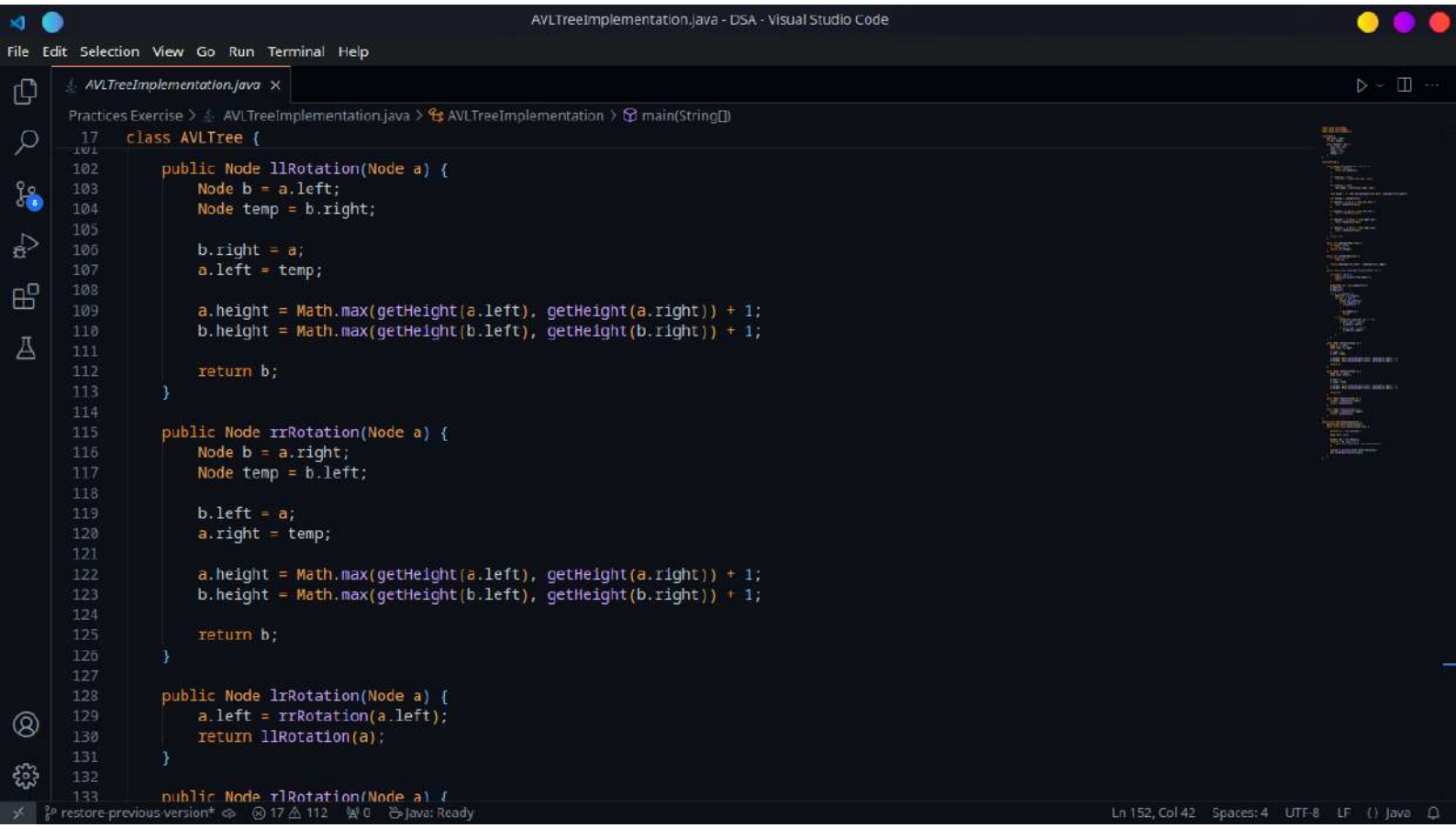
Heigh Balanced.

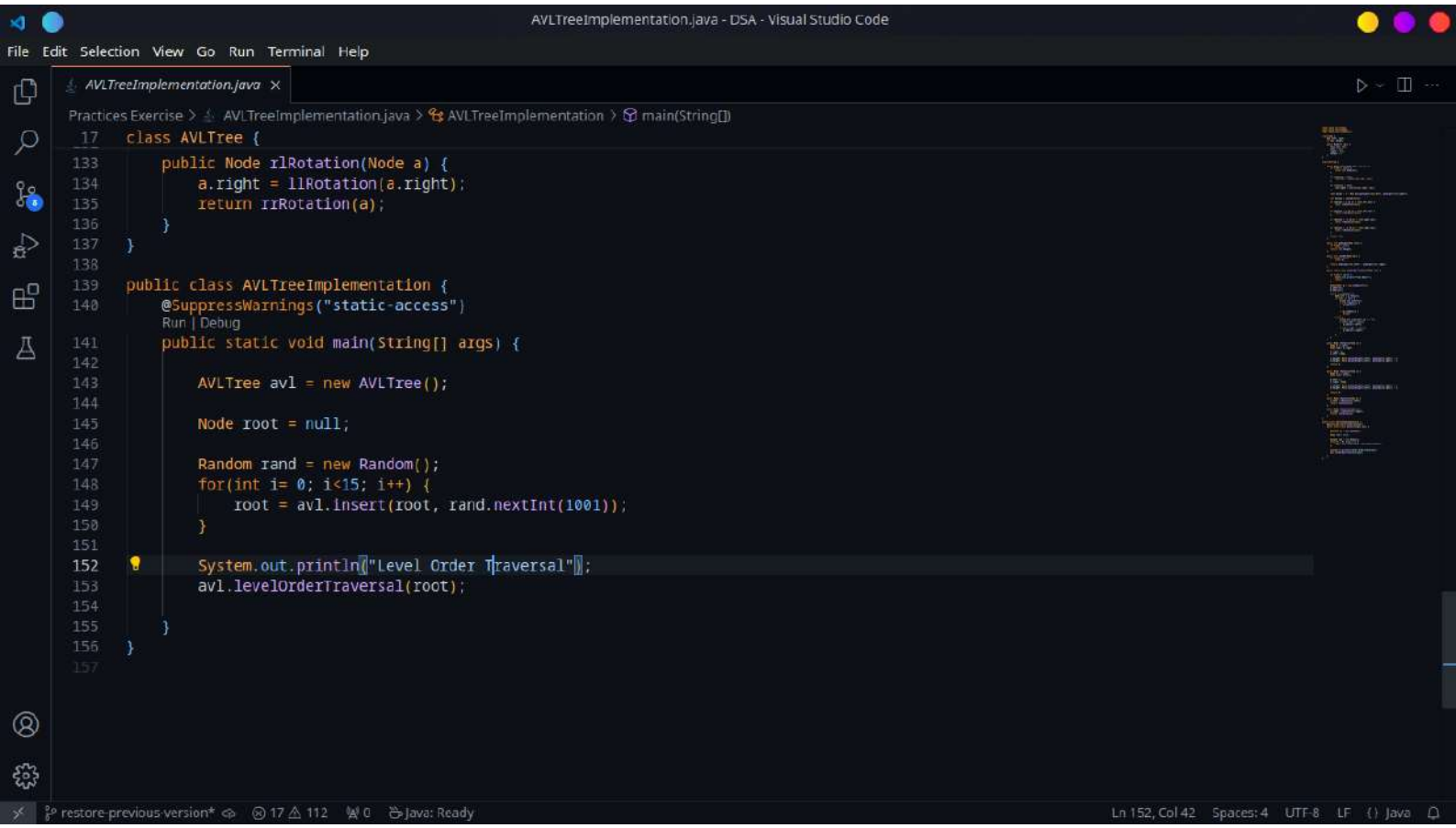

AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
import java.util.Queue;
import java.util.Random;
import java.util.LinkedList;
class Node {
Node left, right;
int val, height;
public Node(int val) {
this.val = val;
left = null;
right = null;
height = 1;
}
}
class AVLTree {
public Node insert(Node root, int val) {
if (root == null) {
return new Node(val);
}
if (root.val > val) {
root.left = insert(root.left, val);
}
if (root.val < val) {
root.right = insert(root.right, val);
}
root.height = 1 + Math.max(getHeight(root.left), getHeight(root.right));
restore-previous-version*
Java: Activating...
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java

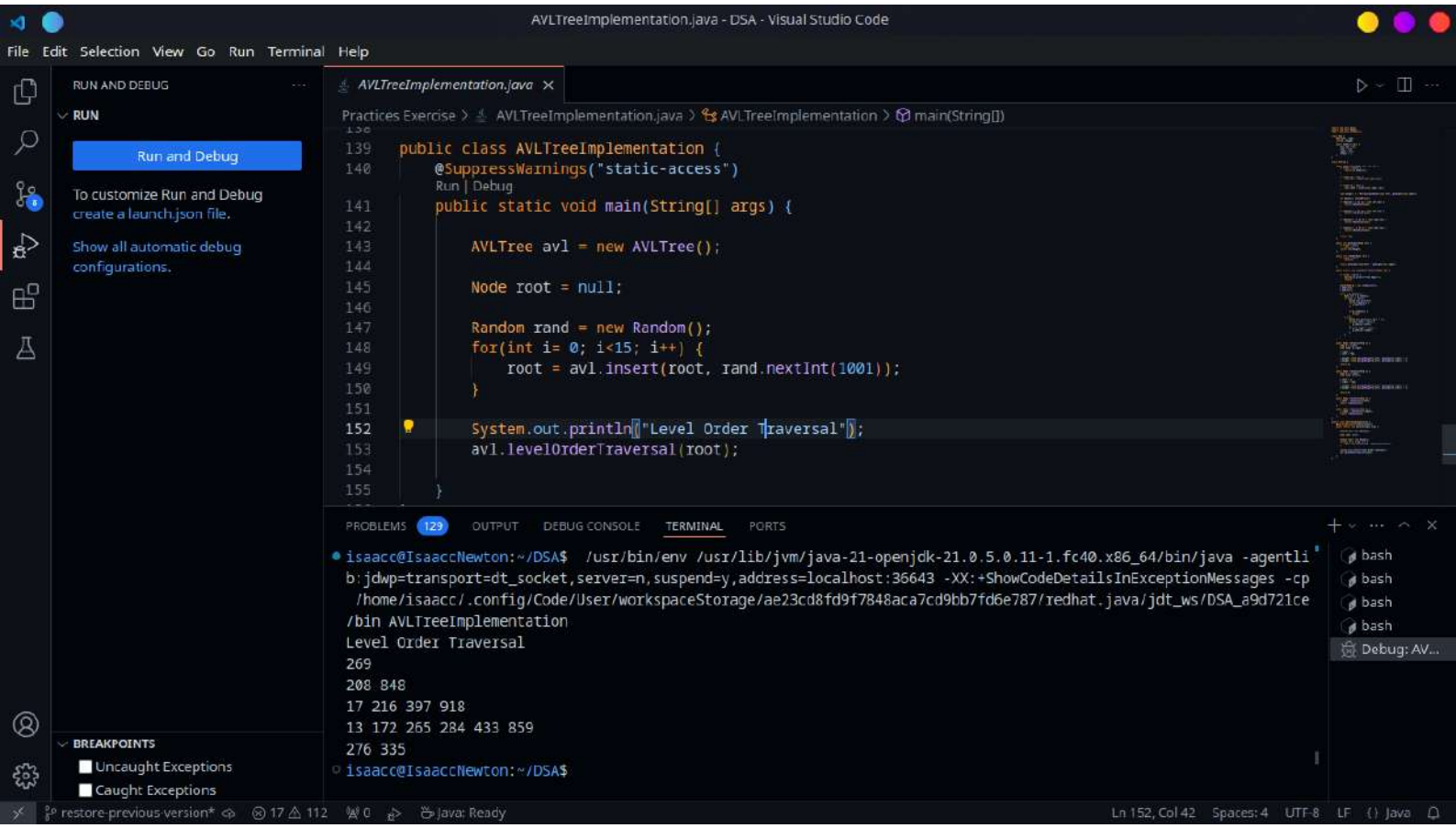

AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
class AVLTree {
public Node insert(Node root, int val) {
if (root.val > val) {
root.left = insert(root.left, val);
}
if (root.val < val) {
root.right = insert(root.right, val);
}
root.height = 1 + Math.max(getHeight(root.left), getHeight(root.right));
int balance = checkBF(root);
if (balance > 1 && val < root.left.val) {
return llRotation(root);
}
if (balance > 1 && val > root.left.val) {
return lrRotation(root);
}
if (balance < -1 && val > root.right.val) {
return rrRotation(root);
}
if (balance < -1 && val < root.right.val) {
return rlRotation(root);
}
return root;
}
restore-previous-version*
17 112
0
Java: Searching... - 80% 6 files to index (/home/isaacc/.m2/repository/org/junit/platform/junit
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java


AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
class AVLTree {
public int getHeight(Node root) {
if (root == null)
return 0;
return root.height;
}
public int checkBF(Node root) {
if (root == null)
return 0;
return getHeight(root.left) - getHeight(root.right);
}
public static void levelOrderTraversal(Node root) {
if (root == null) {
System.out.println("Tree Empty!");
return;
}
Queue<Node> q = new LinkedList<>();
q.add(root);
q.add(null);
while (!q.isEmpty()) {
Node curr = q.remove();
if (curr == null) {
System.out.println();
if (!q.isEmpty()) {
q.add(null);
}
restore-previous-version*
17 112
0
Java: Ready
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java


AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
class AVLTree {
public static void levelOrderTraversal(Node root) {
System.out.println("Tree Empty!");
return;
}
Queue<Node> q = new LinkedList<>();
q.add(root);
q.add(null);
while (!q.isEmpty()) {
Node curr = q.remove();
if (curr == null) {
System.out.println();
if (!q.isEmpty()) {
q.add(null);
}
if (q.isEmpty()) {
break;
}
} else {
System.out.print(curr.val + " ");
if (curr.left != null) {
q.add(curr.left);
}
if (curr.right != null) {
q.add(curr.right);
}
}
}
}
restore-previous-version* 17 112 0 Java: Ready
Ln 152, Col 42 Spaces: 4 UTF-8 LF Java


AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
class AVLTree {
public Node llRotation(Node a) {
Node b = a.left;
Node temp = b.right;
b.right = a;
a.left = temp;
a.height = Math.max(getHeight(a.left), getHeight(a.right)) + 1;
b.height = Math.max(getHeight(b.left), getHeight(b.right)) + 1;
return b;
}
public Node rrRotation(Node a) {
Node b = a.right;
Node temp = b.left;
b.left = a;
a.right = temp;
a.height = Math.max(getHeight(a.left), getHeight(a.right)) + 1;
b.height = Math.max(getHeight(b.left), getHeight(b.right)) + 1;
return b;
}
public Node lrRotation(Node a) {
a.left = rrRotation(a.left);
return llRotation(a);
}
public Node rlRotation(Node a) {
restore-previous-version*
17 112
0
Java: Ready
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java


AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
class AVLTree {
public Node rlRotation(Node a) {
a.right = llRotation(a.right);
return rrRotation(a);
}
}
public class AVLTreeImplementation {
@SuppressWarnings("static-access")
Run | Debug
public static void main(String[] args) {
AVLTree avl = new AVLTree();
Node root = null;
Random rand = new Random();
for(int i= 0; i<15; i++) {
root = avl.insert(root, rand.nextInt(1001));
}
System.out.println("Level Order Traversal");
avl.levelOrderTraversal(root);
}
}
restore-previous-version*
17 112
0
Java: Ready
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java


AVLTreeImplementation.java - DSA - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
AVLTreeImplementation.java
Practices Exercise > AVLTreeImplementation.java > AVLTreeImplementation > main(String[])
public class AVLTreeImplementation {
@SuppressWarnings("static-access")
Run | Debug
public static void main(String[] args) {
AVLTree avl = new AVLTree();
Node root = null;
Random rand = new Random();
for(int i= 0; i<15; i++) {
root = avl.insert(root, rand.nextInt(1001));
}
System.out.println("Level Order Traversal");
avl.levelOrderTraversal(root);
}
PROBLEMS 129
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
isaacc@IsaaccNewton:~/DSA$ /usr/bin/env /usr/lib/jvm/java-21-openjdk-21.0.5.0.11-1.fc40.x86_64/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:36643 -XX:+ShowCodeDetailsInExceptionMessages -cp /home/isaacc/.config/Code/User/workspaceStorage/ae23cd8fd9f7848aca7cd9bb7fd6e787/redhat.java/jdt_ws/DSA_a9d721ce/bin AVLTreeImplementation
Level Order Traversal
269
208 848
17 216 397 918
13 172 265 284 433 859
276 335
isaacc@IsaaccNewton:~/DSA$
bash
bash
bash
bash
Debug: AV...
BREAKPOINTS
Uncaught Exceptions
Caught Exceptions
restore-previous-version*
17 112
0
Java: Ready
Ln 152, Col 42
Spaces: 4
UTF-8
LF
Java