**package** pppp;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** BinaryTree {

//Represent a node of binary tree

**public** **static** **class** Node{

**int** data;

Node left;

Node right;

**public** Node(**int** data){

//Assign data to the new node, set left and right children to null

**this**.data = data;

**this**.left = **null**;

**this**.right = **null**;

}

}

//Represent the root of binary tree

**public** Node root;

**public** BinaryTree(){

root = **null**;

}

//insertNode() will add new node to the binary tree

**public** **void** insertNode(**int** data) {

//Create a new node

Node newNode = **new** Node(data);

//Check whether tree is empty

**if**(root == **null**){

root = newNode;

**return**;

}

**else** {

Queue<Node> queue = **new** LinkedList<Node>();

//Add root to the queue

queue.add(root);

**while**(**true**) {

Node node = queue.remove();

//If node has both left and right child, add both the child to queue

**if**(node.left != **null** && node.right != **null**) {

queue.add(node.left);

queue.add(node.right);

}

**else** {

//If node has no left child, make newNode as left child

**if**(node.left == **null**) {

node.left = newNode;

queue.add(node.left);

}

//If node has left child but no right child, make newNode as right child

**else** {

node.right = newNode;

queue.add(node.right);

}

**break**;

}

}

}

}

//inorder() will perform inorder traversal on binary search tree

**public** **void** inorderTraversal(Node node) {

//Check whether tree is empty

**if**(root == **null**){

System.***out***.println("Tree is empty");

**return**;

}

**else** {

**if**(node.left!= **null**)

inorderTraversal(node.left);

System.***out***.print(node.data + " ");

**if**(node.right!= **null**)

inorderTraversal(node.right);

}

}

**public** **static** **void** main(String[] args) {

BinaryTree bt = **new** BinaryTree();

//Add nodes to the binary tree

bt.insertNode(1);

//1 will become root node of the tree

System.***out***.println("Binary tree after insertion");

//Binary after inserting nodes

bt.inorderTraversal(bt.root);

bt.insertNode(2);

bt.insertNode(3);

//2 will become left child and 3 will become right child of root node 1

System.***out***.println("\nBinary tree after insertion");

//Binary after inserting nodes

bt.inorderTraversal(bt.root);

bt.insertNode(4);

bt.insertNode(5);

//4 will become left child and 5 will become right child of node 2

System.***out***.println("\nBinary tree after insertion");

//Binary after inserting nodes

bt.inorderTraversal(bt.root);

bt.insertNode(6);

bt.insertNode(7);

//6 will become left child and 7 will become right child of node 3

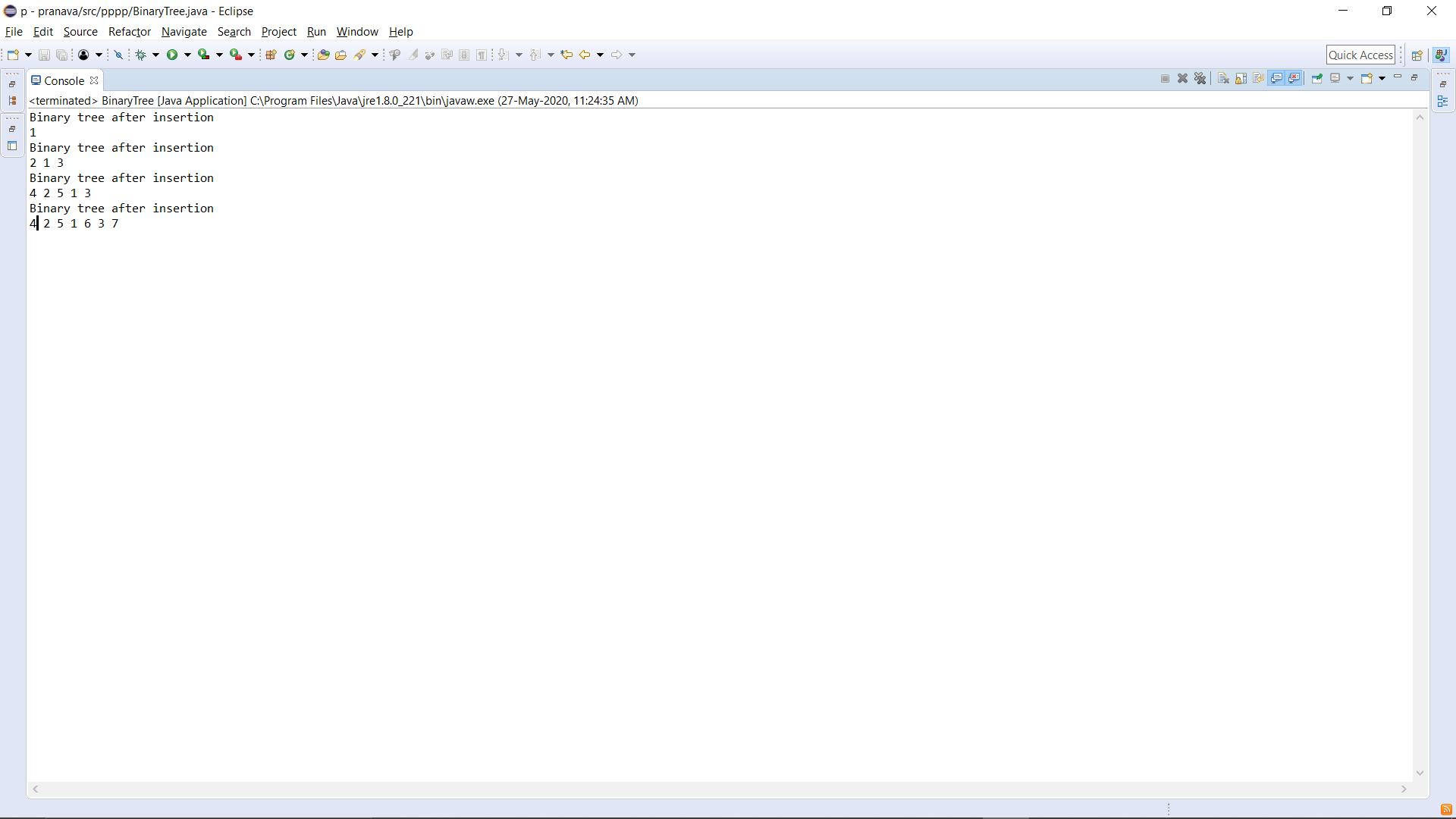
System.***out***.println("\nBinary tree after insertion");

//Binary after inserting nodes

bt.inorderTraversal(bt.root);

}

}

 **OUTPUT:**