```java
// This is my code
// Arizza Santos
// CS451 ASN 5

class Node
{
  protected int num;
  protected BinaryTree left, right;

  // constructor
  public Node(int num)
  {
    this.num = num;
    right = new BinaryTree();
    left = new BinaryTree();
  }

  /*
   * purpose: insert a number
   * input: n - number to insert
   * return: true if number was inserted, false otherwise
   */
  public boolean insert(int n)
  {
    Node nd = new Node(n);

    if (n < num)
    {
      if (left.emptyTree())
        left = new BinaryTree(nd);
      else
        return left.insert(n);
    }
    else
    {
      if (right.emptyTree())
        right = new BinaryTree(nd);
      else
        return right.insert(n);
    }

    return false;
  }

}

class BinaryTree
{
  protected Node root;

  // constructors
  public BinaryTree() { root = null; }
  public BinaryTree(Node n) { root = n; }

  /*
   * purpose: checks if tree is empty
   * input: nothing
   * returns: true if empty, otherwise false
   */
  public boolean emptyTree() { return (root == null); }
```

```java
  /*
   * purpose: insert a number starting from root; delegates to Node class
   * input: n - number to insert
   * return: true if number was inserted, false otherwise
   */
  public boolean insert(int n)
  {
    Node newNode = new Node(n);
    if (root == null)
      root = newNode;
    else
      return root.insert(n);

    return false;
  }

  /*
   * purpose: inorder traversal of the tree
   * input: nothing
   * returns: String of inorder traversal
   */
  public String inorder()
  {
    if (root == null)
      return " ";
    else
      return root.left.inorder() + root.num + root.right.inorder();
  }

  /*
   * purpose: prints inorder traversal of the tree
   * input: nothing
   * returns: nothing - outputs inorder sequence
   */
  public void printInorder() { System.out.println(inorder()); }

  /*
   * purpose: preorder traversal of the tree
   * input: nothing
   * returns: String of preorder traversal
   */
  public String preorder()
  {
    if (root == null)
      return "";
    else return " " + root.num + root.left.preorder() + root.right.preorder();
  }

  /*
   * purpose: prints preorder traversal of the tree
   * input: nothing
   * returns: nothing - outputs preorder sequence
   */
  public void printPreorder() { System.out.println(preorder()); }

  /*
   * purpose: postorder traversal of the tree
   * input: nothing
   * returns: String of postorder traversal
   */
  public String postorder()
  {
    if (root == null)
      return "";
    else return root.left.postorder() + root.right.postorder() + " "
                + root.num;
  }
  /*
   * purpose: prints postorder traversal of the tree
   * input: nothing
   * returns: nothing - outputs postorder sequence
   */
  public void printPostorder() { System.out.println(postorder()); }

  /*
   * purpose: string representation of binary tree object
   * input: nothing
   * returns: String - binary tree
   */
  public String toString()
  {
    if (root == null)
      return "\'null\'";
    else
      return root.num + "(" + root.left.toString() + ")("
             + root.right.toString() + ")";
  }

}

public class ArizzaTree
{
  //purpose: run code
  public static void main(String[] args)
  {
    BinaryTree bt = new BinaryTree();

    bt.insert(7);
    bt.insert(1);
    bt.insert(9);
    bt.insert(0);
    bt.insert(3);
    bt.insert(8);
    bt.insert(10);
    bt.insert(2);
    bt.insert(5);
    bt.insert(4);
    bt.insert(6);

    System.out.println("Binary Tree");
    System.out.println(bt);

    System.out.println("Inorder: ");
    bt.printInorder();
    System.out.println("Preorder: ");
    bt.printPreorder();
    System.out.println("Postorder: ");
    bt.printPostorder();
    System.out.println();

    /* Tests

      // empty tree
      BinaryTree none = new BinaryTree();
      System.out.println(none);

      // tree with one number (root)
      BinaryTree one = new BinaryTree();
      one.insert(7);
      System.out.println(one); // 7 ('null') ('null')

      // tree with one number using a parameterized constructor
      Node n = new Node(7);
      BinaryTree oneNode = new BinaryTree(n);
      System.out.println(oneNode); // 7 ('null') ('null')
```

```java
      // tree with left child only
      BinaryTree withLeft = new BinaryTree();
      withLeft.insert(7);
      withLeft.insert(6);
      System.out.println(withLeft); // 7 (6 ('null') ('null') ) ('null')

      // tree with right child only
      BinaryTree withRight = new BinaryTree();
      withRight.insert(7);
      withRight.insert(8);
      System.out.println(withRight); // 7 ('null') (8 ('null') ('null') )

      // tree with left and right child
      BinaryTree withLR = new BinaryTree();
      withLR.insert(7);
      withLR.insert(6);
      withLR.insert(8);
      System.out.println(withLR); // 7 (6 ('null') ('null') ) (8 ('null') ('null
') )

      // test traversals
      withLR.printInorder(); // 6 7 8
      withLR.printPreorder(); // 7 6 8
      withLR.printPostorder(); // 6 8 7
      */
  }
}
```