```
/**
 * BinaryTree.java.
 * Defines a binary tree data structure.
 *
 * @author     Dean Hendrix (dh@auburn.edu)
 * @version    2017-03-27
 *
 */
public class BinaryTree {

   /**
    * Defines the node structure for the binary tree.
    */
   static class Node {
      Object element;
      Node left;
      Node right;

      public Node(Object e) {
         element = e;
      }
   }

   // the root of this binary tree
   Node root;

   /**
    * Builds a binary tree and calls various methods on it.
    */
   public static void main(String[] args) {
      BinaryTree t = new BinaryTree();
      // root (level 1)
      t.root = new Node("A");

      // level 2
      t.root.left = new Node("B");
      t.root.right = new Node("C");

      // level 3
      t.root.left.left = new Node("D");
      t.root.left.right = new Node("E");
      t.root.right.left = new Node("F");

      // level 4
      t.root.left.left.left = new Node("G");
      t.root.left.left.right = new Node("H");
      t.root.left.right.right = new Node("I");
      t.root.right.left.right = new Node("J");

      System.out.println();
      System.out.println("Preorder: " + t.toPreorderString());
      System.out.println();
      System.out.println("Inorder: " + t.toInorderString());
      System.out.println();
      System.out.println("Postorder: " + t.toPostorderString());
      System.out.println();
      System.out.println("Levelorder: " + t.toLevelorderString());
      System.out.println();
      System.out.println("Height of A: " + t.height(t.root));
   }

   /**
    * Returns a string composed of the node labels of this binary tree listed
    * in preorder.
    */
   String toPreorderString() {
      if (root == null) {
         return "";
      }
      StringBuilder s = new StringBuilder();
```

```
   preorderString(root, s);
   s.delete(s.length() - 2, s.length());
   return s.toString();
}

/**
 * Builds a preorder representation of this binary tree in s.
 */
void preorderString(Node n, StringBuilder s) {
   if (n != null) {
      s.append(n.element + ", ");
      preorderString(n.left, s);
      preorderString(n.right, s);
   }
}

/**
 * Returns a string composed of the node labels of this binary tree listed
 * in inorder.
 */
String toInorderString() {
   if (root == null) {
      return "";
   }
   StringBuilder s = new StringBuilder();
   inorderString(root, s);
   s.delete(s.length() - 2, s.length());
   return s.toString();
}

/**
 * Builds an inorder representation of this binary tree in s.
 */
void inorderString(Node n, StringBuilder s) {
   if (n != null) {
      inorderString(n.left, s);
      s.append(n.element + ", ");
      inorderString(n.right, s);
   }
}

/**
 * Returns a string composed of the node labels of this binary tree listed
 * in postorder.
 */
String toPostorderString() {
   if (root == null) {
      return "";
   }
   StringBuilder s = new StringBuilder();
   postorderString(root, s);
   s.delete(s.length() - 2, s.length());
   return s.toString();
}

/**
 * Builds a postorder representation of this binary tree in s.
 */
void postorderString(Node n, StringBuilder s) {
   if (n != null) {
      postorderString(n.left, s);
      postorderString(n.right, s);
      s.append(n.element + ", ");
   }
}

/**
 * Returns a string composed of the node labels of this binary tree listed
 * in level order.
 */
String toLevelorderString() {
```

```
      if (root == null) {
         return "";
      }
      StringBuilder s = new StringBuilder();
      levelorderString(root, s);
      s.delete(s.length() - 2, s.length());
      return s.toString();
   }

   /**
    * Builds a level order representation of this binary tree in s.
    */
   void levelorderString(Node r, StringBuilder s) {
      java.util.Deque<Node> q = new java.util.ArrayDeque<Node>();
      q.add(r);
      while (!q.isEmpty()) {
         Node n = q.remove();
         s.append(n.element + ", ");
         if (n.left != null) {
            q.add(n.left);
         }
         if (n.right != null) {
            q.add(n.right);
         }
      }
   }

   /**
    * Returns the height of node n in this binary tree.
    */
   int height(Node n) {
      if (n == null) {
         return 0;
      }
      int leftHeight = height(n.left);
      int rightHeight = height(n.right);
      return 1 + Math.max(leftHeight, rightHeight);
   }


}
```