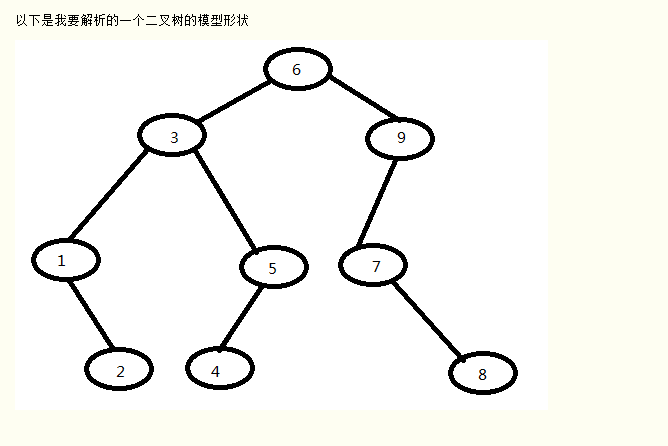
二叉树的前序遍历，中序遍历，后序遍历（递归的写法）



///////////////////////////////////递归////////////////////////////////////////////////////////

//下面先构建树的类

**package** facehandjava.tree;  
  
**public class** Node {  
 **private int data**;  
 **private** Node **leftNode**;  
 **private** Node **rightNode**;  
 **public** Node(**int** data, Node leftNode, Node rightNode){  
 **this**.**data** = data;  
 **this**.**leftNode** = leftNode;  
 **this**.**rightNode** = rightNode;  
 }  
  
 **public int** getData() {  
 **return data**;  
 }  
 **public void** setData(**int** data) {  
 **this**.**data** = data;  
 }  
 **public** Node getLeftNode() {  
 **return leftNode**;  
 }  
 **public void** setLeftNode(Node leftNode) {  
 **this**.**leftNode** = leftNode;  
 }  
 **public** Node getRightNode() {  
 **return rightNode**;  
 }  
 **public void** setRightNode(Node rightNode) {  
 **this**.**rightNode** = rightNode;  
 }  
  
}

**package** facehandjava.tree;  
  
**public class** RecursiveBinaryTree {  
 */\*\*  
 \*  
 \* 二叉树的先序中序后序排序（递归版本）  
 \*/* **public** Node init() {*//注意必须逆序建立，先建立子节点，再逆序往上建立，因为非叶子结点会使用到下面的节点，而初始化是按顺序初始化的，不逆序建立会报错* Node J = **new** Node(8, **null**, **null**);  
 Node H = **new** Node(4, **null**, **null**);  
 Node G = **new** Node(2, **null**, **null**);  
 Node F = **new** Node(7, **null**, J);  
 Node E = **new** Node(5, H, **null**);  
 Node D = **new** Node(1, **null**, G);  
 Node C = **new** Node(9, F, **null**);  
 Node B = **new** Node(3, D, E);  
 Node A = **new** Node(6, B, C);  
 **return** A; *//返回根节点* }  
  
 **public static void** main(String[] args) {  
 RecursiveBinaryTree tree = **new** RecursiveBinaryTree();  
 Node root = tree.init();  
 System.***out***.println(**"先序遍历(递归)"**);  
 tree.FirstTraversal(root);  
 System.***out***.println(**""**);  
 System.***out***.println(**"中序遍历(递归)"**);  
 tree.MiddleTraversal(root);  
 System.***out***.println(**""**);  
 System.***out***.println(**"后序遍历(递归)"**);  
 tree.LastTraversal(root);  
 System.***out***.println(**""**);  
 }  
  
 **private void** FirstTraversal(Node root) {  
 System.***out***.print(root.getData());  
 **if** (root.getLeftNode() != **null**) {  
 FirstTraversal(root.getLeftNode());  
 }  
 **if** (root.getRightNode() != **null**) {  
 FirstTraversal(root.getRightNode());  
 }  
 }  
  
 **private void** MiddleTraversal(Node root) {  
 **if** (root.getLeftNode() != **null**) {  
 MiddleTraversal(root.getLeftNode());  
 }  
 System.***out***.print(root.getData());  
 **if** (root.getRightNode() != **null**) {  
 MiddleTraversal(root.getRightNode());  
 }  
 }  
  
 **private void** LastTraversal(Node root) {  
 **if** (root.getLeftNode() != **null**) {  
 LastTraversal(root.getLeftNode());  
 }  
 **if** (root.getRightNode() != **null**) {  
 LastTraversal(root.getRightNode());  
 }  
 System.***out***.print(root.getData());  
 }  
  
}

///////////////////////////////////非递归////////////////////////////////////////////////////////

//下面先构建树的类

**package** facehandjava.tree;  
  
**public class** Node {  
 **private int data**;  
 **private** Node **leftNode**;  
 **private** Node **rightNode**;  
 **public** Node(**int** data, Node leftNode, Node rightNode){  
 **this**.**data** = data;  
 **this**.**leftNode** = leftNode;  
 **this**.**rightNode** = rightNode;  
 }  
  
 **public int** getData() {  
 **return data**;  
 }  
 **public void** setData(**int** data) {  
 **this**.**data** = data;  
 }  
 **public** Node getLeftNode() {  
 **return leftNode**;  
 }  
 **public void** setLeftNode(Node leftNode) {  
 **this**.**leftNode** = leftNode;  
 }  
 **public** Node getRightNode() {  
 **return rightNode**;  
 }  
 **public void** setRightNode(Node rightNode) {  
 **this**.**rightNode** = rightNode;  
 }  
  
}

**package** facehandjava.tree;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Stack;  
  
  
**public class** NonRecursiveBinaryTree {  
 */\*\*  
 \* 二叉树的先序中序后序排序（非递归版本）  
 \*/* **public** Node init() {*//注意必须逆序建立，先建立子节点，再逆序往上建立，因为非叶子结点会使用到下面的节点，而初始化是按顺序初始化的，不逆序建立会报错* Node J = **new** Node(8, **null**, **null**);  
 Node H = **new** Node(4, **null**, **null**);  
 Node G = **new** Node(2, **null**, **null**);  
 Node F = **new** Node(7, **null**, J);  
 Node E = **new** Node(5, H, **null**);  
 Node D = **new** Node(1, **null**, G);  
 Node C = **new** Node(9, F, **null**);  
 Node B = **new** Node(3, D, E);  
 Node A = **new** Node(6, B, C);  
 **return** A; *//返回根节点* }  
 **public static void** main(String[] args) {  
 NonRecursiveBinaryTree tree = **new** NonRecursiveBinaryTree();  
 Node root = tree.init();  
 System.***out***.println(**"先序遍历(应该)\n"** +  
 **"631254978\n"** +  
 **"中序遍历(应该)\n"** +  
 **"123456789\n"** +  
 **"后序遍历(应该)\n"** +  
 **"214538796"**);  
 System.***out***.println(**"先序遍历(非递归)"**);  
 *FirstTraversal\_Stack*(root);  
 System.***out***.println(**""**);  
 System.***out***.println(**"中序遍历(非递归)"**);  
 *MiddleTraversal\_Stack*(root);  
 System.***out***.println(**""**);  
 System.***out***.println(**"后序遍历(非递归)"**);  
 *LastTraversal\_Stack*(root);  
 System.***out***.println(**""**);  
 }  
  
 **private static void** FirstTraversal\_Stack(Node root) {  
 Stack<Node> stack = **new** Stack<>();  
 **while** (root != **null** || !stack.isEmpty()) {  
 **if** (root != **null**) {  
 System.***out***.print(root.getData());  
 stack.push(root);  
 root = root.getLeftNode();  
 } **else** {  
 root = stack.pop();  
 root = root.getRightNode();  
 }  
 }  
 }  
  
 **private static void** MiddleTraversal\_Stack(Node root) {  
 Stack<Node> stack = **new** Stack<>();  
 **while** (root != **null** || !stack.isEmpty()) {  
 **if** (root!= **null**) {  
 stack.push(root);  
 root = root.getLeftNode();  
 } **else** {  
 root = stack.pop();  
 System.***out***.print(root.getData());  
 root = root.getRightNode();  
 }  
 }  
 }  
  
 **private static void** LastTraversal\_Stack(Node root) {  
 Stack<Node> stack = **new** Stack<>();  
 Stack<Node> out = **new** Stack<>();  
 **while** (root != **null** || !stack.isEmpty()) {  
 **if** (root != **null**) {  
 out.push(root);  
 stack.push(root);  
 root = root.getRightNode();  
 } **else** {  
 root = stack.pop();  
 root = root.getLeftNode();  
 }  
 }  
 **while** (!out.isEmpty()) {  
 System.***out***.print(out.pop().getData());  
 }  
 }  
}