1.二叉树（遍历、查找、删除）

public class BinaryTreeDemo *{* public static void main*(*String*[]* args*) {* BinaryTree binaryTree = new BinaryTree*()*;  
 P05\_HeroNode root = new P05\_HeroNode*(*1, "宋江"*)*;  
 P05\_HeroNode node2 = new P05\_HeroNode*(*2, "卢俊义"*)*;  
 P05\_HeroNode node3 = new P05\_HeroNode*(*3, "吴用"*)*;  
 P05\_HeroNode node4 = new P05\_HeroNode*(*4, "林冲"*)*;  
 P05\_HeroNode node5 = new P05\_HeroNode*(*5, "蔡啊包"*)*;  
  
  
 root.setLeft*(*node2*)*;*//root.left = node2;* root.setRight*(*node3*)*;  
 node3.setRight*(*node4*)*;  
 binaryTree.setRoot*(*root*)*;  
 node3.setLeft*(*node5*)*;  
  
 System.*out*.println*(*"前序"*)*;*//1,2,3,5,4* binaryTree.perOrder*()*;  
  
 System.*out*.println*(*"中序"*)*;*//2,1,5,3,4* binaryTree.infixOrder*()*;  
  
 System.*out*.println*(*"后序"*)*;*//2,5,4,3,1* binaryTree.pastOrder*()*;  
 System.*out*.println*(*" "*)*;  
  
 *//前序遍历查找指定节点，并输出次数  
 //System.out.println("前序遍历查找 :");* P05\_HeroNode resNode1 = binaryTree.preOrderSearch*(*5*)*;  
 if*(*resNode1 != null*){* System.*out*.println*(*resNode1.getNo*()*+resNode1.getName*())*;  
 *}*else*{* System.*out*.println*(*"没有找到"*)*;  
 *}* System.*out*.println*(*" "*)*;  
  
 *//中序遍历查找指定节点，并输出次数  
 //System.out.println("中序遍历查找 :");* P05\_HeroNode resNode2 = binaryTree.infixOrderSearch*(*5*)*;  
 if*(*resNode2 != null*){* System.*out*.println*(*resNode2.getNo*()*+resNode2.getName*())*;  
 *}*else*{* System.*out*.println*(*"没有找到"*)*;  
 *}* System.*out*.println*(*" "*)*;  
  
 *//后序遍历查找指定节点，并输出次数  
 //System.out.println("后序遍历查找 :");* P05\_HeroNode resNode3 = binaryTree.postOrderSearch*(*5*)*;  
 if*(*resNode3 != null*){* System.*out*.println*(*resNode3.getNo*()*+resNode3.getName*())*;  
 *}*else*{* System.*out*.println*(*"没有找到"*)*;  
 *}* System.*out*.println*(*" "*)*;  
  
 System.*out*.println*(*"删除前"*)*;  
 binaryTree.perOrder*()*;*//1,2,3,5,4* binaryTree.delNode*(*3*)*;*//删除子数* System.*out*.println*(*"删除后"*)*;  
 binaryTree.perOrder*()*;*//1,2  
  
 }  
}  
  
//创建二叉树*class BinaryTree *{* private P05\_HeroNode root;  
  
 public void setRoot*(*P05\_HeroNode root*) {* this.root = root;  
 *}  
  
 //前序遍历* public void perOrder*() {* if *(*this.root != null*) {* this.root.preOrder*()*;  
 *}* else *{* System.*out*.println*(*"二叉树为空"*)*;  
 *}  
 }  
  
 //中序遍历* public void infixOrder*() {* if *(*this.root != null*) {* this.root.infixOrder*()*;  
 *}* else *{* System.*out*.println*(*"二叉树为空"*)*;  
 *}  
 }  
  
 //后序遍历* public void pastOrder*() {* if *(*this.root != null*) {* this.root.pastOrder*()*;  
 *}* else *{* System.*out*.println*(*"二叉树为空"*)*;  
 *}  
 }  
  
 //中序遍历查找* public P05\_HeroNode infixOrderSearch*(*int no*) {* if *(*this.root != null*) {* return this.root.infixOrderSearch*(*no*)*;  
 *}* else *{* return null;  
 *}  
 }  
  
 //中序遍历查找* public P05\_HeroNode postOrderSearch*(*int no*) {* if *(*this.root != null*) {* return this.root.postOrderSearch*(*no*)*;  
 *}* else *{* return null;  
 *}  
 }  
  
 //前序遍历查找* public P05\_HeroNode preOrderSearch*(*int no*) {* if *(*this.root != null*) {* return this.root.preOrderSearch*(*no*)*;  
 *}* else *{* return null;  
 *}  
 }  
  
 //如果只有一个 root 结点, 这里立即判断 root 是不是就是要删除结点* public void delNode*(*int no*){* if*(*root != null*){* if*(*root.getNo*()* == no*){* root =null;  
 *}*else*{* root.delNode*(*no*)*;  
 *}  
 }*else*{* System.*out*.println*(*"空树，不能删除"*)*;  
 *}  
 }  
}  
  
//创建节点*class P05\_HeroNode *{* private int no;  
 private String name;  
 private P05\_HeroNode left; *//默认为null* private P05\_HeroNode right;  
  
 public P05\_HeroNode*(*int no, String name*) {* this.no = no;  
 this.name = name;  
 *}* public int getNo*() {* return no;  
 *}* public String getName*() {* return name;  
 *}* public P05\_HeroNode getLeft*() {* return left; *}* public P05\_HeroNode getRight*() {* return right;  
 *}* public void setNo*(*int no*) {* this.no = no;  
 *}* public void setName*(*String name*) {* this.name = name;  
 *}* public void setLeft*(*P05\_HeroNode left*) {* this.left = left;  
 *}* public void setRight*(*P05\_HeroNode right*) {* this.right = right;  
 *}* @Override  
 public String toString*() {* return "HeroNode{" +  
 "no=" + no +  
 ", name='" + name + '\'' +  
 ", left=" + left +  
 ", right=" + right +  
 '}';  
 *}  
  
 //前序遍历* public void preOrder*() {* System.*out*.println*(*this*)*;  
 if *(*this.left != null*) {* this.left.preOrder*()*;  
 *}* if *(*this.right != null*) {* this.right.preOrder*()*;  
 *}  
 }  
  
 //中序遍历* public void infixOrder*() {* if *(*this.left != null*) {* this.left.infixOrder*()*;  
 *}* System.*out*.println*(*this*)*;  
 if *(*this.right != null*) {* this.right.infixOrder*()*;  
 *}  
 }  
  
 //后序遍历* public void pastOrder*() {* if *(*this.left != null*) {* this.left.pastOrder*()*;  
 *}* if *(*this.right != null*) {* this.right.pastOrder*()*;  
 *}* System.*out*.println*(*this*)*;  
 *}  
  
 //前序遍历查找  
 /\*  
 如果找到返回No，没找到返回null  
 \*/* public P05\_HeroNode preOrderSearch*(*int no*){* System.*out*.println*(*"进入遍历"*)*;  
 *//上面这句用来index查找次数，要注意放置这个sout的位置，否则会多余* if*(*this.no == no*){* return this;  
 *}* P05\_HeroNode resNode = null;*//用来存储* if*(*this.left != null*){* resNode = this.left.preOrderSearch*(*no*)*;  
 *}* if*(*resNode != null*){//说明左子树找到了* return resNode;  
 *}* if*(*this.right != null*){* resNode = this.right.preOrderSearch*(*no*)*;  
 *}* return resNode;*//可能找到，也可能还是null  
 }  
  
 //中序遍历查找* public P05\_HeroNode infixOrderSearch*(*int no*){* P05\_HeroNode resNode = null;  
 if*(*this.left != null*){* resNode = this.left.infixOrderSearch*(*no*)*;  
 *}* if*(*resNode != null*){//说明左子树找到了* return resNode;  
 *}* System.*out*.println*(*"进入遍历"*)*;  
 if*(*this.no == no*){* return this;  
 *}* if*(*this.right != null*){* resNode = this.right.infixOrderSearch*(*no*)*;  
 *}* return resNode;*//可能找到，也可能还是null  
 }  
  
 //后序遍历查找* public P05\_HeroNode postOrderSearch*(*int no*){* P05\_HeroNode resNode = null;  
  
 if*(*this.left != null*){* resNode = this.left.postOrderSearch*(*no*)*;  
 *}* if*(*resNode != null*){//说明左子树找到了* return resNode;  
 *}* if*(*this.right != null*){* resNode = this.right.postOrderSearch*(*no*)*;  
 *}* if*(*resNode != null*){//说明左子树找到了* return resNode;  
 *}* System.*out*.println*(*"进入遍历"*)*;  
 if*(*this.no == no*){* return this;  
 *}* return resNode;*//可能找到，也可能还是null  
 }  
  
 //遍历—删除节点* public void delNode*(*int no*){  
 //2. 如果当前结点的左子结点不为空，并且左子结点 就是要删除结点，就将 this.left = null; 并且就返回(结束递归删除)* if*(*this.left != null && this.left.no == no*) {* this.left = null; return;  
 *}  
  
 //3.如果当前结点的右子结点不为空，并且右子结点 就是要删除结点，就将 this.right= null ;并且就返回(结束递归删除)* if*(*this.right != null && this.right.no == no*) {* this.right = null; return;  
 *}  
  
 //4.我们就需要向左子树进行递归删除* if*(*this.left != null*) {* this.left.delNode*(*no*)*;  
 *//不return，因为向左不一定就可以成功  
 }  
  
 //5.则应当向右子树进行递归删除* if*(*this.right != null*) {* this.right.delNode*(*no*)*;  
 *}  
  
 }  
  
}  
// This code is contributed by Lance Cai*

2.二叉树遍历

class Node{

public int no; //data域

public Node left; //左节点

public Node right; //右节点

public Node(int no) {

this.no = no;

}

//前序遍历 按照：中左右遍历

public void preShow(){

if (this==null){

System.out.println("该树为空!");

return; //方法执行结束

}

System.out.print(this.no+" "); //输出节点

if(this.left!=null){//遍历左节点

this.left.preShow();

}

if (this.right!=null){ //遍历右节点

this.right.preShow();

}

}

//中序遍历 按照:左中右遍历

public void infixShow(){

if (this==null){

System.out.println("该树为空!");

return;

}

if (this.left!=null){ //遍历左节点

this.left.infixShow();

}

System.out.print(this.no+" ");

if (this.right!=null){ //遍历右节点

this.right.infixShow();

}

}

//后序遍历 按照：左右中遍历

public void postShow(){

if (this==null){

System.out.println("该树为空!");

return;

}

if (this.left!=null){

this.left.postShow();

}

if (this.right!=null){

this.right.postShow();

}

System.out.print(this.no+" ");

}

}

\*\*测试代码：\*\*

public static void main(String[] args) {

Node n1=new Node(1);

Node n2=new Node(2);

Node n3=new Node(3);

Node n4=new Node(4);

Node n5=new Node(5);

Node n6=new Node(6);

Node n7=new Node(7);

n1.left=n2;

n1.right=n3;

n2.left=n4;

n2.right=n5;

n3.left=n6;

n3.right=n7; //通过指向形成二叉树

n1.preShow(); //前序遍历

System.out.println("");

n1.infixShow(); //中序遍历

System.out.println("");

n1.postShow(); //后序遍历

}