```java
class BinaryTreeTraversal {
        /**
         * @param root 树根节点
         * 递归先序遍历
         */
        public static void preOrderRec(Node root){
                if(root!=null){
                        System.out.println(root.value);
                        preOrderRec(root.left);
                        preOrderRec(root.right);
                }
        }
        /**
         * @param root 树根节点
         * 递归中序遍历
         */
        public static void inOrderRec(Node root){
                if(root!=null){
                        preOrderRec(root.left);
                        System.out.println(root.value);
                        preOrderRec(root.right);
                }
        }
        /**
         * @param root 树根节点
         * 递归后序遍历
         */
        public static void postOrderRec(Node root){
                if(root!=null){
                        preOrderRec(root.left);
                        preOrderRec(root.right);
                        System.out.println(root.value);
                }
        }
```

```java
/**
 *
 * @param root 树根节点
 * 利用栈实现循环先序遍历二叉树
 * 这种实现类似于图的深度优先遍历（DFS）
 * 维护一个栈，将根节点入栈，然后只要栈不为空，出栈并访问，接着依次将访问节点的右节
 * 这种方式应该是对先序遍历的一种特殊实现（看上去简单明了），但是不具备很好的扩展性
 */
public static void preOrderStack_1(Node root){
        if(root==null)return;
        Stack<Node> s=new Stack<Node>();
        s.push(root);
        while(!s.isEmpty()){
                Node temp=s.pop();
                System.out.println(temp.value);
                if(temp.right!=null) s.push(temp.right);
                if(temp.left!=null) s.push(temp.left);
        }
}
/**
 *
 * @param root 树的根节点
 * 利用栈模拟递归过程实现循环先序遍历二叉树
 * 这种方式具备扩展性，它模拟递归的过程，将左子树点不断的压入栈，直到null，然后处理
 */
public static void preOrderStack_2(Node root){
        if(root==null)return;
        Stack<Node> s=new Stack<Node>();
        while(root!=null||!s.isEmpty()){
                while(root!=null){
                        System.out.println(root.value);
                        s.push(root);//先访问再入栈
                        root=root.left;
                }
```

```
84          */
85          public static void preOrderStack_2(Node root){
86                  if(root==null)return;
87                  Stack<Node> s=new Stack<Node>();
88                  while(root!=null||!s.isEmpty()){
89                          while(root!=null){
90                                  System.out.println(root.value);
91                                  s.push(root);//先访问再入栈
92                                  root=root.left;
93                          }
94                          root=s.pop();
95                          root=root.right;//如果是null，出栈并处理右子树
96                  }
97          }
98          /**
```

```
     * @param root 树根节点
     * 利用栈模拟递归过程实现循环中序遍历二叉树
     * 思想和上面的preOrderStack_2相同，只是访问的时间是在左子树都处理完直到null的时候
     */
    public static void inOrderStack(Node root){
        if(root==null)return;
        Stack<Node> s=new Stack<Node>();
        while(root!=null||!s.isEmpty()){
            while(root!=null){
                s.push(root);//先访问再入栈
                root=root.left;
            }
            root=s.pop();
            System.out.println(root.value);
            root=root.right;//如果是null，出栈并处理右子树
        }
    }
    /**
     *
     * @param root 树根节点
     * 后序遍历不同于先序和中序，它是要先处理完左右子树，然后再处理根(回溯)，所以需要一
     */
    public static void postOrderStack(Node root){
        if(root==null)return;
        Stack<Node> s=new Stack<Node>();
        Map<Node,Boolean> map=new HashMap<Node,Boolean>();
        s.push(root);
        while(!s.isEmpty()){
            Node temp=s.peek();
            if(temp.left!=null&&!map.containsKey(temp.left)){
                temp=temp.left;
                while(temp!=null){
            if(map.containsKey(temp))break;
            else s.push(temp);
            temp=temp.left;
```

```
                temp=temp.left;
                while(temp!=null){
            if(map.containsKey(temp))break;
            else s.push(temp);
            temp=temp.left;
        }
                continue;
            }
            if(temp.right!=null&&!map.containsKey(temp.right)){
        s.push(temp.right);
        continue;
    }
            Node t=s.pop();
            map.put(t,true);
            System.out.println(t.value);
        }
    }
    /**
     *
     * @param root 树根节点
     * 层序遍历二叉树，用队列实现，先将根节点入队列，只要队列不为空，然后出队列，并访问，
     */
    public static void levelTravel(Node root){
        if(root==null)return;
        Queue<Node> q=new LinkedList<Node>();
        q.add(root);
        while(!q.isEmpty()){
            Node temp =  q.poll();
            System.out.println(temp.value);
            if(temp.left!=null)q.add(temp.left);
            if(temp.right!=null)q.add(temp.right);
        }
    }
}
```

后序遍历的思想：

```java
        public static ArrayList postOrder1(TreeNode root){
                ArrayList alist = new ArrayList();
                Stack<TreeNode> stack = new Stack<TreeNode>();
                if(root == null)
                        return alist;
                TreeNode cur,pre = null;
                stack.push(root);
                while(!stack.empty()){
                        cur = stack.peek();
                        if((cur.left == null && cur.right == null) || (pre != null &&
                                TreeNode temp = stack.pop();
                                alist.add(temp.val);
                                pre = temp;
                        }
                        else{
                                if(cur.right != null)
                                        stack.push(cur.right);
                                if(cur.left != null)
                                        stack.push(cur.left);
                        }
                }
                return alist;
        }
```