三，面向对象

一）基础

- Go仅支持封装，不支持继承和多态
- Go没有class，只有struct

- 只有使用指针才可以改变结构体内容
- nil指针也可以调用方法
- 要改变内容必须使用指针接收者
- 结构过大也考虑使用指针接收者
- 一致性：如有指针接收者，最好都是指针接收者

```
package tree

type treeNode struct {
  value       int
  left, right *treeNode
}

// 自定义工厂函数
// 返回局部变量地址，仍然可以给其他方法使用
func createTreeNode(value int) *treeNode {
  return &treeNode{value: value}
}

func (node *treeNode) print() {
  fmt.Print(node.value, " ")
}

// 等同于下面的
/*func print(node treeNode) {
  fmt.Print(node.value)
}*/

// (node treeNode)相当于其他语言的this
// 中序遍历
func (node *treeNode) loopTree() {
  if node == nil {
    return
  }
  node.left.loopTree()
  node.print()
  node.right.loopTree()
}

func (node *treeNode) setValue(value int) {
```

```
  if node == nil {
    fmt.Println("node is nil")
    return
  }
  node.value = value
}

func main() {
  var root treeNode
  root = treeNode{
    value: 3,
  }
  root.left = &treeNode{}
  root.right = &treeNode{5, nil, nil}
  root.right.left = new(treeNode)
  root.left.right = createTreeNode(2)
  root.right.left.setValue(4)

  root.loopTree()
}
```

- 值接收者是go语言特有

二）封装

- 名字一般使用CamelCase
- 首字母大写代表public
- 首字母小写代表private

三）包

- 每个目录一个包
- main包包含可执行入口
- 为结构定义的方法必须放在同一个包内
- 可以是不同文件

四）扩展已有类型

```
package main

import "xxxxx/tree"

// 扩展上面的treeNode
type MyTreeNode struct {
  node *tree.TreeNode
}

func (myNode *myTreeNode) postOrder() {
  if myNode == nil || myNode.node == nil {
    return
```

```
  }
  left := myTreeNode{myNode.node.Left}.postOrder
  right := myTreeNode{myNode.node.Right}.postOrder
  myNode.node.Print()
}
```

eg:实现一个队列

```
// queue.go
package queue

type Queue []int

func (q *Queue) Push(v int) {
  *q = append(*q, v)
}

func (q *Queue) Pop() int {
  head := (*q)[0]
  *q = (*q)[1:]
  return head
}

func (q *Queue) IsEmpty() bool {
  return len(*q) == 0
}
```