实验编号： 2 **四川师大《IOS》实验报告 2018** 年 **9** 月 **12** 日

### **计算机科学学院** 2016 级 4 班 实验名称： 闭包、扩展、泛型、协议 \_

姓名：\_蒋宇童\_\_\_\_\_\_\_\_ 学号：\_2016110419\_\_\_\_\_\_\_\_\_ 指导老师：\_\_李贵洋\_\_ 实验成绩:\_\_\_\_\_

**实验\_二\_ \_\_\_\_\_\_**闭包、扩展、泛型、协议**\_\_\_\_\_\_\_\_**

1. 实验目的
2. 掌握闭包、扩展、泛型的定义;
3. 掌握排序等采用闭包方法的实现；
4. 掌握类的派生、协议；
5. 掌握版本控制git的进阶使用；
6. 实验内容

* 作业1（闭包、扩展、泛型）：

1. 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;
2. 给定一个String数组，用filter函数选出能被转成Int的字符串
3. 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔
4. 用 reduce 方法一次求出整数数组的最大值、最小值、总数和
5. 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；
6. 扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；
7. 实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。
8. 掌握版本控制git的进阶使用
   1. git reset 恢复到之前修改的版本；
   2. git log 看提交记录；
   3. git branch 新建分支；
   4. git checkout 切换分支；
   5. git branch -d 删除分支；

* 作业2:（枚举、类、派生、协议）(红色字体为新增内容)

1. 实现Person类：
   1. 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
   2. 具有指定构造函数和便利构造函数；
   3. 两个Person实例对象可以用==和!=进行比较；
   4. Person实例可以直接用print输出；
   5. Person增加run方法(方法里面直接print输出Person XXX is running;
2. 从Person分别派生Teacher类和Student类：
   1. Teacher类增加属性title，实例可以直接用print输出；
   2. Student类增加属性stuNo，实例可以直接用print输出；
   3. Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)
3. 分别构造多个Person、Teacher和Student对象，并将这些对象存入同一个数组中；
4. 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；
5. 修改Teacher和Student，让这两个类实现该协议；
6. 对数组执行以下要求：
   1. 分别统计Person、Teacher和Student对象的个数并放入一字典中，统计完后输出字典内容；
   2. 对数组按以下要求排序并输出：age、fullName、gender+age；
   3. 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；
7. 实验主要流程、基本操作或核心代码、算法片段（该部分如不够填写，请另加附页）

* 作业1（闭包、扩展、泛型）：

1. 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;
2. 给定一个String数组，用filter函数选出能被转成Int的字符串
3. 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔
4. 用 reduce 方法一次求出整数数组的最大值、最小值、总数和
5. 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；
6. 扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；
7. 实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。
8. 掌握版本控制git的进阶使用
   1. git reset 恢复到之前修改的版本；
   2. git log 看提交记录；
   3. git branch 新建分支；
   4. git checkout 切换分支；
   5. git branch -d 删除分支；

* 程序代码：

import UIKit

//（1） 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;

var dirctionary:[[String:Any]]=[["name":"蒋宇童","age":20],["name":"蒋开心","age":30],["name":"蒋快乐","age":40]]

var arr=dirctionary.map{$0["age"]}

//（2） 给定一个String数组，用filter函数选出能被转成Int的字符串

var filterInt=["1","2","3","开心","每一天"]

var arr2=filterInt.filter{Int($0) != nil}

print(arr2.count)

//（3） 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔

var arr3=["蒋","宇","童","很","开心"]

var co=arr3.count

var n=0

var arr4=arr3.reduce("") {

if n<co-1

{

n=n+1

return $0+$1+","

}

else

{

n=n+1

return $0+$1

}

}

print(arr4)

var arr5=[1,2,3,4,5,6,7,8]

var arr6=[1,2,3,4,56,]

let tuple = arr6.reduce((max: arr6[0], min:arr6[0], sum: 0)){ (max($0.max, $1),min($0.min, $1), $0.sum + $1) }

print(tuple)

// 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；

func one()->Void

{

print("返回值为空")

}

func two()->Int

{

return 2

}

func three(max:Int)->Int

{

return 3

}

func four(max:Int)->Void

{

print("返回值为空，参为整数")

}

var arr7:[Any]=[one,two,three,four]

for (key,value) in arr7.enumerated()

{

if value is (Int)->Int

{

print("下标为\(key)的函数是参数为整数，返回值也是整数")

}

}

//扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；

extension Int {

func Sqrt() -> Double {

return sqrt(Double(self))

}

}

var m=4.Sqrt()

print("4的方数为\(m)")

//实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。

func found<T:Comparable>(a:T...)->(T,T)

{

return a.reduce((max:a[0],min:a[0]), {(max($0.max,$1),min($0.min,$1))})

}

print(found(a:1,2,3,4,5,6,77))

print(found(a:1.1,2,3,4,5,6,77.9))

print(found(a:"jiang","yu","tong","hen","kai","xing"))

/\*

（1） 实现Person类：

a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；

b) 具有指定构造函数和便利构造函数；

c) 两个Person实例对象可以用==和!=进行比较；

d) Person实例可以直接用print输出；

e) Person增加run方法(方法里面直接print输出Person XXX is running;

\*/

enum Gender

{

case male

case female

}

class Person:CustomStringConvertible

{

var firstName:String

var lastName:String

var age:Int

var gender:Gender

init(firstName:String, lastName:String,age:Int,gender:Gender) {

self.firstName=firstName

self.lastName=lastName

self.age=age

self.gender=gender

}

var fullName:String

{

get

{

return firstName+lastName

}

}

convenience init(name:String,age:Int)

{

self.init(firstName:name,lastName:"",age:age,gender:Gender.female)

}

static func ==(p1:Person,p2:Person)->Bool

{

return p1.fullName==p2.fullName&&p1.age==p2.age

}

static func != (p1:Person,p2:Person)->Bool

{

return p1.fullName != p2.fullName||p1.age != p2.age

}

var description: String {

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"

}

func run ()->Void

{

print("Person \(fullName) is running;")

}

}

var per1=Person(firstName: "1", lastName: "2", age: 3, gender: .female)

per1.run()

print(per1)

/\*

（2） 从Person分别派生Teacher类和Student类：

a) Teacher类增加属性title，实例可以直接用print输出；

b) Student类增加属性stuNo，实例可以直接用print输出；

c) Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)

（5） 修改Teacher和Student，让这两个类实现该协议；

\*/

/\*

（4） 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；

\*/

enum ProtocolEnum

{

case happy

case lucky

}

protocol SelfWrite

{

var dapartment:ProtocolEnum{get}//自读

func lendBook()

}

//老师

class Teacher:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .happy

}

func lendBook() {

print("我借了8本书")

}

var title:String

init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {

self.title=title

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"

}

override func run() {

print("Teacher \(fullName) is running")

}

}

var t1=Teacher(firstName: "jiang", lastName: "yu", age: 20, gender: .female, title: "good")

t1.run()

//学生

class Student:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .lucky

}

func lendBook() {

print("我很厉害看了超多书")

}

var stuNo:String

init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {

self.stuNo=stuNo

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"

}

override func run() {

print("Student \(fullName) is running")

}

}

var s1=Student(firstName: "jiang", lastName: "yu", age: 20, gender: .female, stuNo: "good")

s1.run()

/\*

c) 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；

\*/

var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)

var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)

var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")

var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")

var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")

var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")

var arry=[p1,p2,tea1,tea2,stu1,stu2]

for i in arry

{

i.run()

if let a = i as? Student

{

a.lendBook()

}

else if let a=i as? Teacher

{

a.lendBook()

}

else

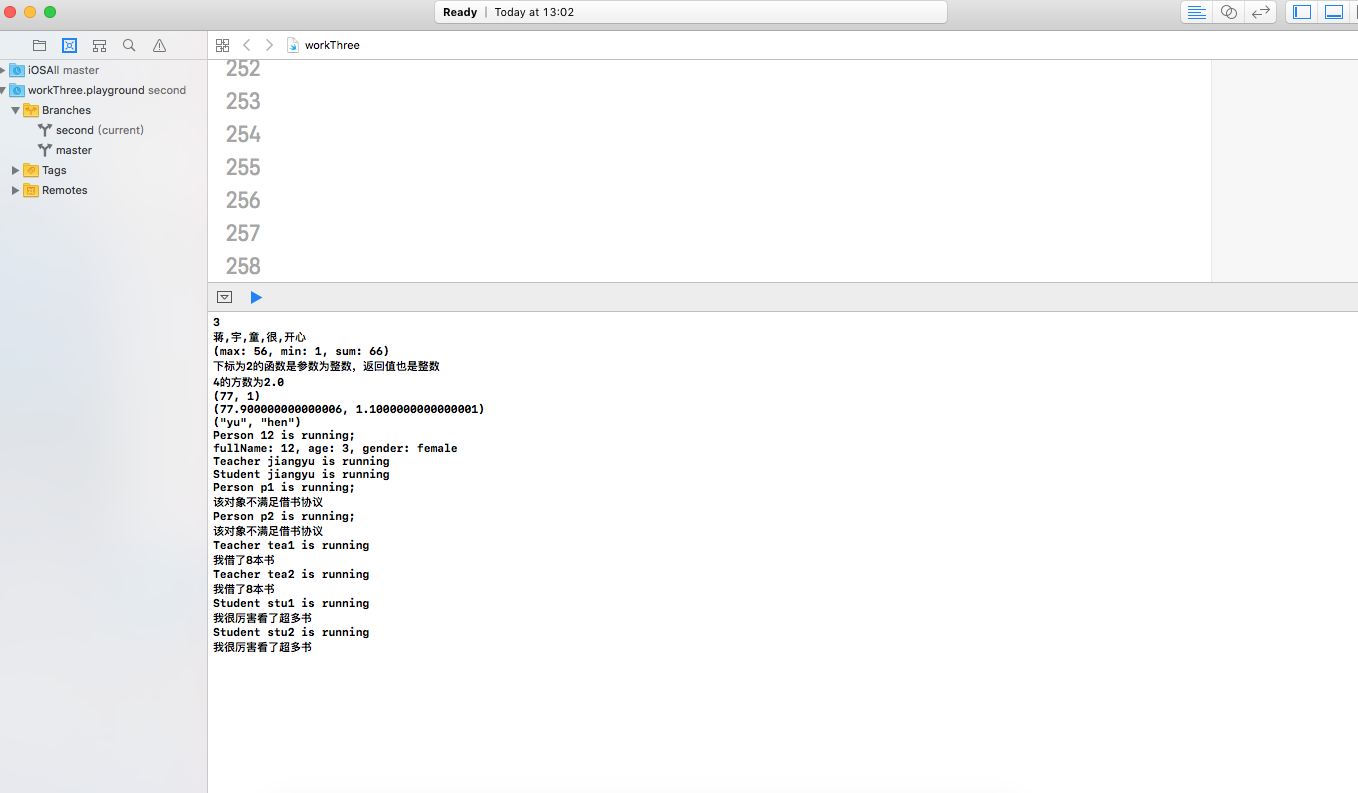
{

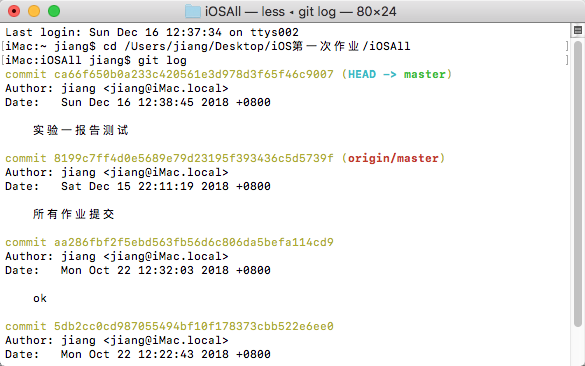
print("该对象不满足借书协议")

}

}

* 运行结果：

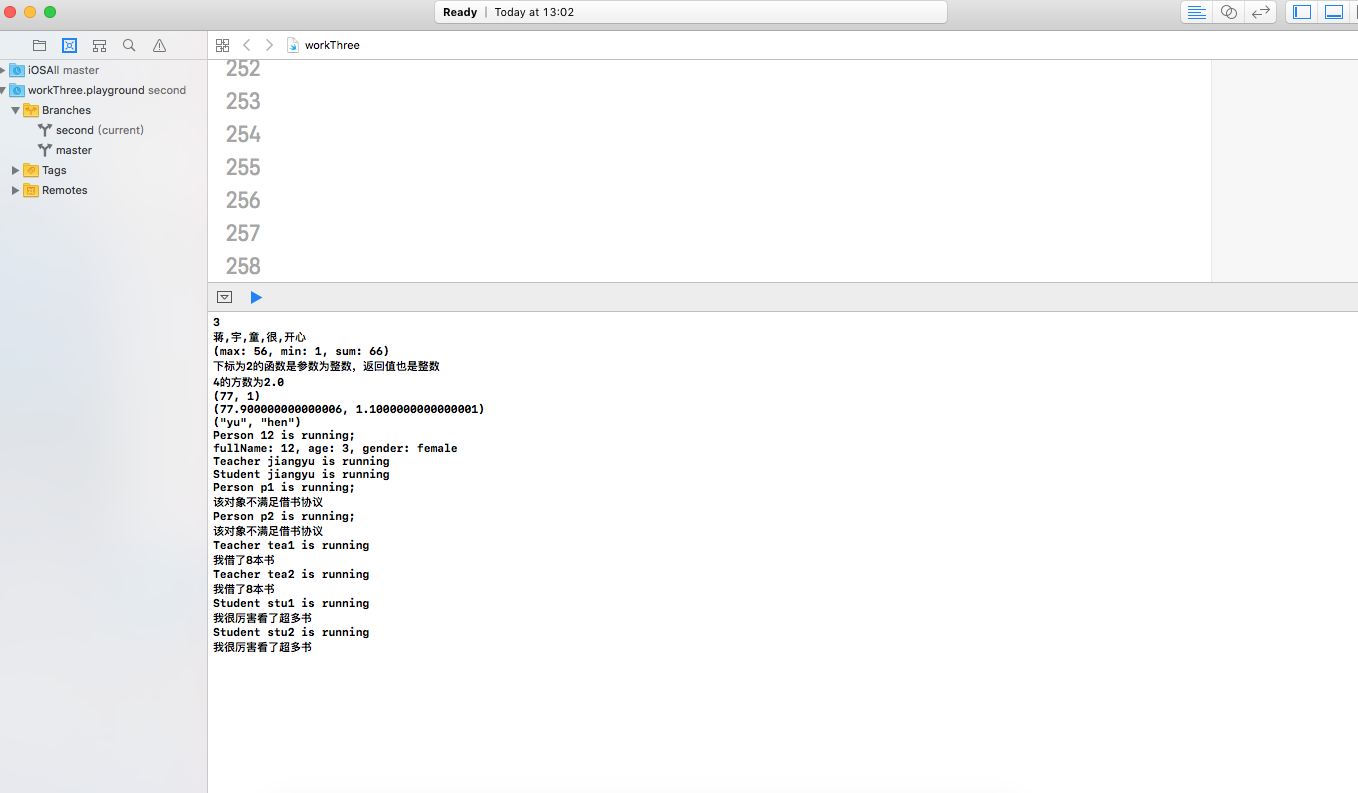




* 作业2:（枚举、类、派生、协议）(红色字体为新增内容)

1. 实现Person类：
   1. 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
   2. 具有指定构造函数和便利构造函数；
   3. 两个Person实例对象可以用==和!=进行比较；
   4. Person实例可以直接用print输出；
   5. Person增加run方法(方法里面直接print输出Person XXX is running;

* 程序代码：
* import UIKit
* //（1） 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;
* var dirctionary:[[String:Any]]=[["name":"蒋宇童","age":20],["name":"蒋开心","age":30],["name":"蒋快乐","age":40]]
* var arr=dirctionary.map{$0["age"]}
* //（2） 给定一个String数组，用filter函数选出能被转成Int的字符串
* var filterInt=["1","2","3","开心","每一天"]
* var arr2=filterInt.filter{Int($0) != nil}
* print(arr2.count)
* //（3） 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔
* var arr3=["蒋","宇","童","很","开心"]
* var co=arr3.count
* var n=0
* var arr4=arr3.reduce("") {
* if n<co-1
* {
* n=n+1
* return $0+$1+","
* }
* else
* {
* n=n+1
* return $0+$1
* }
* }
* print(arr4)
* var arr5=[1,2,3,4,5,6,7,8]
* var arr6=[1,2,3,4,56,]
* let tuple = arr6.reduce((max: arr6[0], min:arr6[0], sum: 0)){ (max($0.max, $1),min($0.min, $1), $0.sum + $1) }
* print(tuple)
* // 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；
* func one()->Void
* {
* print("返回值为空")
* }
* func two()->Int
* {
* return 2
* }
* func three(max:Int)->Int
* {
* return 3
* }
* func four(max:Int)->Void
* {
* print("返回值为空，参为整数")
* }
* var arr7:[Any]=[one,two,three,four]
* for (key,value) in arr7.enumerated()
* {
* if value is (Int)->Int
* {
* print("下标为\(key)的函数是参数为整数，返回值也是整数")
* }
* }
* //扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；
* extension Int {
* func Sqrt() -> Double {
* return sqrt(Double(self))
* }
* }
* var m=4.Sqrt()
* print("4的方数为\(m)")
* //实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。
* func found<T:Comparable>(a:T...)->(T,T)
* {
* return a.reduce((max:a[0],min:a[0]), {(max($0.max,$1),min($0.min,$1))})
* }
* print(found(a:1,2,3,4,5,6,77))
* print(found(a:1.1,2,3,4,5,6,77.9))
* print(found(a:"jiang","yu","tong","hen","kai","xing"))
* /\*
* （1） 实现Person类：
* a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
* b) 具有指定构造函数和便利构造函数；
* c) 两个Person实例对象可以用==和!=进行比较；
* d) Person实例可以直接用print输出；
* e) Person增加run方法(方法里面直接print输出Person XXX is running;
* \*/
* enum Gender
* {
* case male
* case female
* }
* class Person:CustomStringConvertible
* {
* var firstName:String
* var lastName:String
* var age:Int
* var gender:Gender
* init(firstName:String, lastName:String,age:Int,gender:Gender) {
* self.firstName=firstName
* self.lastName=lastName
* self.age=age
* self.gender=gender
* }
* var fullName:String
* {
* get
* {
* return firstName+lastName
* }
* }
* convenience init(name:String,age:Int)
* {
* self.init(firstName:name,lastName:"",age:age,gender:Gender.female)
* }
* static func ==(p1:Person,p2:Person)->Bool
* {
* return p1.fullName==p2.fullName&&p1.age==p2.age
* }
* static func != (p1:Person,p2:Person)->Bool
* {
* return p1.fullName != p2.fullName||p1.age != p2.age
* }
* var description: String {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"
* }
* func run ()->Void
* {
* print("Person \(fullName) is running;")
* }
* }
* var per1=Person(firstName: "1", lastName: "2", age: 3, gender: .female)
* per1.run()
* print(per1)
* /\*
* （2） 从Person分别派生Teacher类和Student类：
* a) Teacher类增加属性title，实例可以直接用print输出；
* b) Student类增加属性stuNo，实例可以直接用print输出；
* c) Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)
* （5） 修改Teacher和Student，让这两个类实现该协议；
* \*/
* /\*
* （4） 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；
* \*/
* enum ProtocolEnum
* {
* case happy
* case lucky
* }
* protocol SelfWrite
* {
* var dapartment:ProtocolEnum{get}//自读
* func lendBook()
* }
* //老师
* class Teacher:Person,SelfWrite
* {
* var dapartment: ProtocolEnum
* {
* return .happy
* }
* func lendBook() {
* print("我借了8本书")
* }
* var title:String
* init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {
* self.title=title
* super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)
* }
* override var description: String
* {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"
* }
* override func run() {
* print("Teacher \(fullName) is running")
* }
* }
* var t1=Teacher(firstName: "jiang", lastName: "yu", age: 20, gender: .female, title: "good")
* t1.run()
* //学生
* class Student:Person,SelfWrite
* {
* var dapartment: ProtocolEnum
* {
* return .lucky
* }
* func lendBook() {
* print("我很厉害看了超多书")
* }
* var stuNo:String
* init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {
* self.stuNo=stuNo
* super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)
* }
* override var description: String
* {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"
* }
* override func run() {
* print("Student \(fullName) is running")
* }
* }
* var s1=Student(firstName: "jiang", lastName: "yu", age: 20, gender: .female, stuNo: "good")
* s1.run()
* /\*
* c) 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；
* \*/
* var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)
* var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)
* var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")
* var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")
* var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")
* var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")
* var arry=[p1,p2,tea1,tea2,stu1,stu2]
* for i in arry
* {
* i.run()
* if let a = i as? Student
* {
* a.lendBook()
* }
* else if let a=i as? Teacher
* {
* a.lendBook()
* }
* else
* {
* print("该对象不满足借书协议")
* }
* }
* 运行结果：



1. 从Person分别派生Teacher类和Student类：
   1. Teacher类增加属性title，实例可以直接用print输出；
   2. Student类增加属性stuNo，实例可以直接用print输出；
   3. Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)

* 程序代码：

import UIKit

//（1） 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;

var dirctionary:[[String:Any]]=[["name":"蒋宇童","age":20],["name":"蒋开心","age":30],["name":"蒋快乐","age":40]]

var arr=dirctionary.map{$0["age"]}

//（2） 给定一个String数组，用filter函数选出能被转成Int的字符串

var filterInt=["1","2","3","开心","每一天"]

var arr2=filterInt.filter{Int($0) != nil}

print(arr2.count)

//（3） 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔

var arr3=["蒋","宇","童","很","开心"]

var co=arr3.count

var n=0

var arr4=arr3.reduce("") {

if n<co-1

{

n=n+1

return $0+$1+","

}

else

{

n=n+1

return $0+$1

}

}

print(arr4)

var arr5=[1,2,3,4,5,6,7,8]

var arr6=[1,2,3,4,56,]

let tuple = arr6.reduce((max: arr6[0], min:arr6[0], sum: 0)){ (max($0.max, $1),min($0.min, $1), $0.sum + $1) }

print(tuple)

// 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；

func one()->Void

{

print("返回值为空")

}

func two()->Int

{

return 2

}

func three(max:Int)->Int

{

return 3

}

func four(max:Int)->Void

{

print("返回值为空，参为整数")

}

var arr7:[Any]=[one,two,three,four]

for (key,value) in arr7.enumerated()

{

if value is (Int)->Int

{

print("下标为\(key)的函数是参数为整数，返回值也是整数")

}

}

//扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；

extension Int {

func Sqrt() -> Double {

return sqrt(Double(self))

}

}

var m=4.Sqrt()

print("4的方数为\(m)")

//实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。

func found<T:Comparable>(a:T...)->(T,T)

{

return a.reduce((max:a[0],min:a[0]), {(max($0.max,$1),min($0.min,$1))})

}

print(found(a:1,2,3,4,5,6,77))

print(found(a:1.1,2,3,4,5,6,77.9))

print(found(a:"jiang","yu","tong","hen","kai","xing"))

/\*

（1） 实现Person类：

a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；

b) 具有指定构造函数和便利构造函数；

c) 两个Person实例对象可以用==和!=进行比较；

d) Person实例可以直接用print输出；

e) Person增加run方法(方法里面直接print输出Person XXX is running;

\*/

enum Gender

{

case male

case female

}

class Person:CustomStringConvertible

{

var firstName:String

var lastName:String

var age:Int

var gender:Gender

init(firstName:String, lastName:String,age:Int,gender:Gender) {

self.firstName=firstName

self.lastName=lastName

self.age=age

self.gender=gender

}

var fullName:String

{

get

{

return firstName+lastName

}

}

convenience init(name:String,age:Int)

{

self.init(firstName:name,lastName:"",age:age,gender:Gender.female)

}

static func ==(p1:Person,p2:Person)->Bool

{

return p1.fullName==p2.fullName&&p1.age==p2.age

}

static func != (p1:Person,p2:Person)->Bool

{

return p1.fullName != p2.fullName||p1.age != p2.age

}

var description: String {

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"

}

func run ()->Void

{

print("Person \(fullName) is running;")

}

}

var per1=Person(firstName: "1", lastName: "2", age: 3, gender: .female)

per1.run()

print(per1)

/\*

（2） 从Person分别派生Teacher类和Student类：

a) Teacher类增加属性title，实例可以直接用print输出；

b) Student类增加属性stuNo，实例可以直接用print输出；

c) Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)

（5） 修改Teacher和Student，让这两个类实现该协议；

\*/

/\*

（4） 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；

\*/

enum ProtocolEnum

{

case happy

case lucky

}

protocol SelfWrite

{

var dapartment:ProtocolEnum{get}//自读

func lendBook()

}

//老师

class Teacher:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .happy

}

func lendBook() {

print("我借了8本书")

}

var title:String

init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {

self.title=title

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"

}

override func run() {

print("Teacher \(fullName) is running")

}

}

var t1=Teacher(firstName: "jiang", lastName: "yu", age: 20, gender: .female, title: "good")

t1.run()

//学生

class Student:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .lucky

}

func lendBook() {

print("我很厉害看了超多书")

}

var stuNo:String

init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {

self.stuNo=stuNo

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"

}

override func run() {

print("Student \(fullName) is running")

}

}

var s1=Student(firstName: "jiang", lastName: "yu", age: 20, gender: .female, stuNo: "good")

s1.run()

/\*

c) 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；

\*/

var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)

var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)

var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")

var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")

var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")

var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")

var arry=[p1,p2,tea1,tea2,stu1,stu2]

for i in arry

{

i.run()

if let a = i as? Student

{

a.lendBook()

}

else if let a=i as? Teacher

{

a.lendBook()

}

else

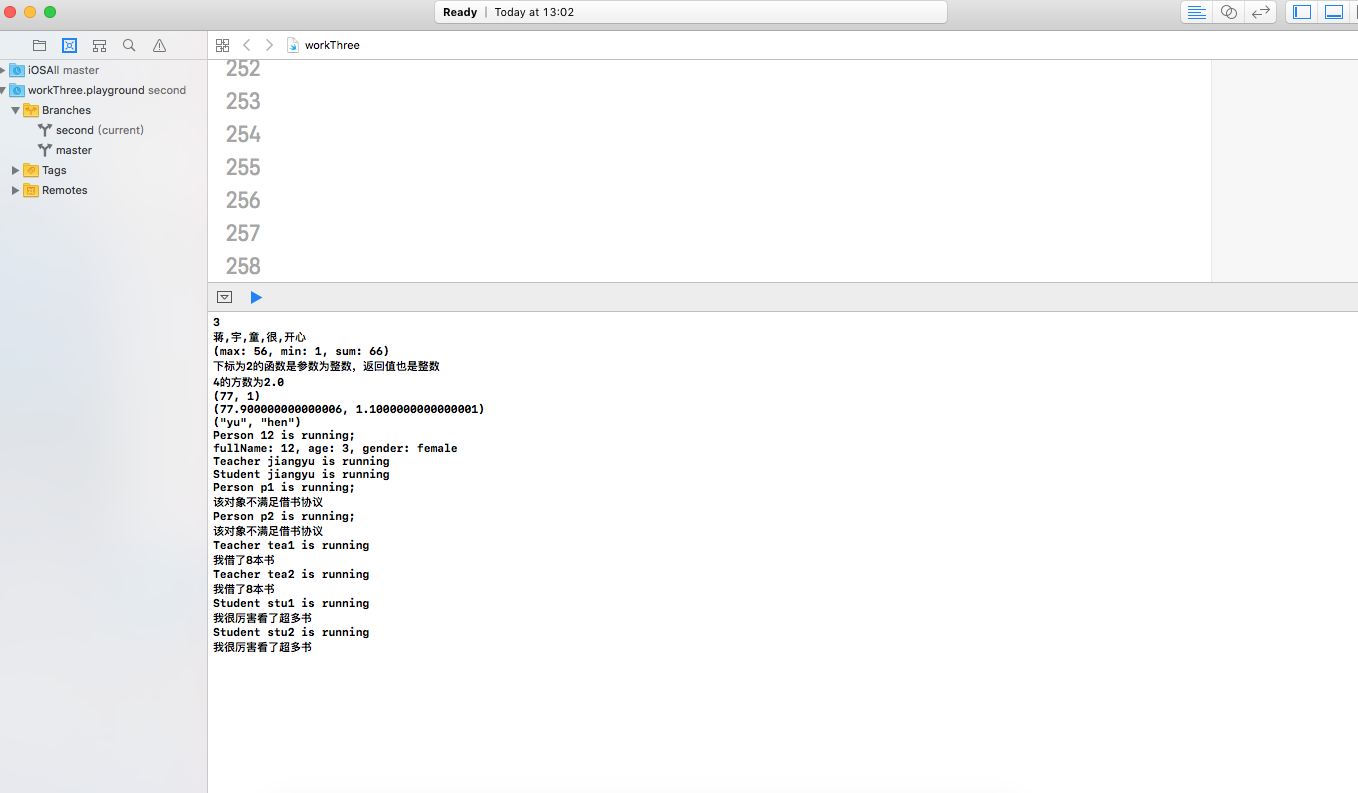
{

print("该对象不满足借书协议")

}

}

* 运行结果：



1. 分别构造多个Person、Teacher和Student对象，并将这些对象存入同一个数组中；

* 程序代码：

import UIKit

//（1） 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;

var dirctionary:[[String:Any]]=[["name":"蒋宇童","age":20],["name":"蒋开心","age":30],["name":"蒋快乐","age":40]]

var arr=dirctionary.map{$0["age"]}

//（2） 给定一个String数组，用filter函数选出能被转成Int的字符串

var filterInt=["1","2","3","开心","每一天"]

var arr2=filterInt.filter{Int($0) != nil}

print(arr2.count)

//（3） 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔

var arr3=["蒋","宇","童","很","开心"]

var co=arr3.count

var n=0

var arr4=arr3.reduce("") {

if n<co-1

{

n=n+1

return $0+$1+","

}

else

{

n=n+1

return $0+$1

}

}

print(arr4)

var arr5=[1,2,3,4,5,6,7,8]

var arr6=[1,2,3,4,56,]

let tuple = arr6.reduce((max: arr6[0], min:arr6[0], sum: 0)){ (max($0.max, $1),min($0.min, $1), $0.sum + $1) }

print(tuple)

// 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；

func one()->Void

{

print("返回值为空")

}

func two()->Int

{

return 2

}

func three(max:Int)->Int

{

return 3

}

func four(max:Int)->Void

{

print("返回值为空，参为整数")

}

var arr7:[Any]=[one,two,three,four]

for (key,value) in arr7.enumerated()

{

if value is (Int)->Int

{

print("下标为\(key)的函数是参数为整数，返回值也是整数")

}

}

//扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；

extension Int {

func Sqrt() -> Double {

return sqrt(Double(self))

}

}

var m=4.Sqrt()

print("4的方数为\(m)")

//实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。

func found<T:Comparable>(a:T...)->(T,T)

{

return a.reduce((max:a[0],min:a[0]), {(max($0.max,$1),min($0.min,$1))})

}

print(found(a:1,2,3,4,5,6,77))

print(found(a:1.1,2,3,4,5,6,77.9))

print(found(a:"jiang","yu","tong","hen","kai","xing"))

/\*

（1） 实现Person类：

a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；

b) 具有指定构造函数和便利构造函数；

c) 两个Person实例对象可以用==和!=进行比较；

d) Person实例可以直接用print输出；

e) Person增加run方法(方法里面直接print输出Person XXX is running;

\*/

enum Gender

{

case male

case female

}

class Person:CustomStringConvertible

{

var firstName:String

var lastName:String

var age:Int

var gender:Gender

init(firstName:String, lastName:String,age:Int,gender:Gender) {

self.firstName=firstName

self.lastName=lastName

self.age=age

self.gender=gender

}

var fullName:String

{

get

{

return firstName+lastName

}

}

convenience init(name:String,age:Int)

{

self.init(firstName:name,lastName:"",age:age,gender:Gender.female)

}

static func ==(p1:Person,p2:Person)->Bool

{

return p1.fullName==p2.fullName&&p1.age==p2.age

}

static func != (p1:Person,p2:Person)->Bool

{

return p1.fullName != p2.fullName||p1.age != p2.age

}

var description: String {

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"

}

func run ()->Void

{

print("Person \(fullName) is running;")

}

}

var per1=Person(firstName: "1", lastName: "2", age: 3, gender: .female)

per1.run()

print(per1)

/\*

（2） 从Person分别派生Teacher类和Student类：

a) Teacher类增加属性title，实例可以直接用print输出；

b) Student类增加属性stuNo，实例可以直接用print输出；

c) Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)

（5） 修改Teacher和Student，让这两个类实现该协议；

\*/

/\*

（4） 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；

\*/

enum ProtocolEnum

{

case happy

case lucky

}

protocol SelfWrite

{

var dapartment:ProtocolEnum{get}//自读

func lendBook()

}

//老师

class Teacher:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .happy

}

func lendBook() {

print("我借了8本书")

}

var title:String

init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {

self.title=title

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"

}

override func run() {

print("Teacher \(fullName) is running")

}

}

var t1=Teacher(firstName: "jiang", lastName: "yu", age: 20, gender: .female, title: "good")

t1.run()

//学生

class Student:Person,SelfWrite

{

var dapartment: ProtocolEnum

{

return .lucky

}

func lendBook() {

print("我很厉害看了超多书")

}

var stuNo:String

init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {

self.stuNo=stuNo

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"

}

override func run() {

print("Student \(fullName) is running")

}

}

var s1=Student(firstName: "jiang", lastName: "yu", age: 20, gender: .female, stuNo: "good")

s1.run()

/\*

c) 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；

\*/

var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)

var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)

var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")

var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")

var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")

var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")

var arry=[p1,p2,tea1,tea2,stu1,stu2]

for i in arry

{

i.run()

if let a = i as? Student

{

a.lendBook()

}

else if let a=i as? Teacher

{

a.lendBook()

}

else

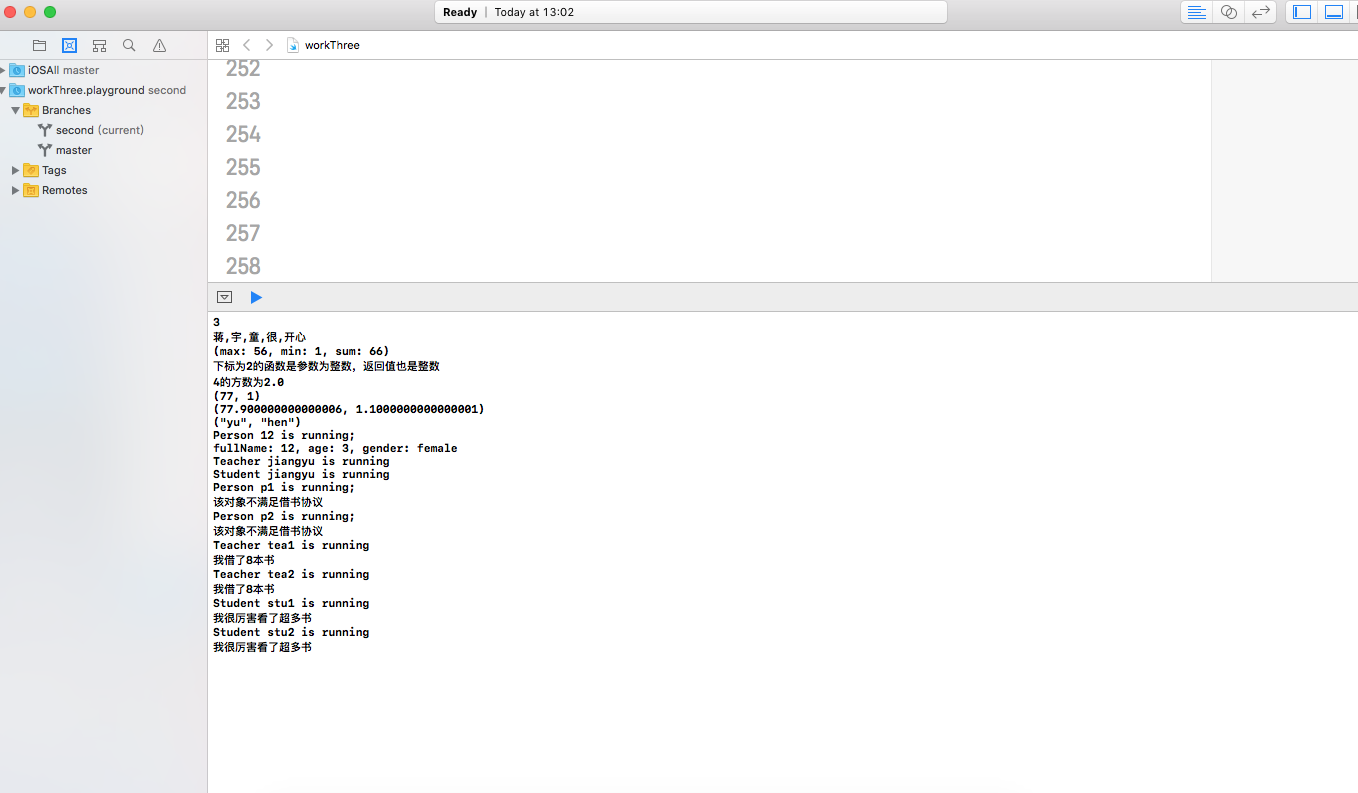
{

print("该对象不满足借书协议")

}

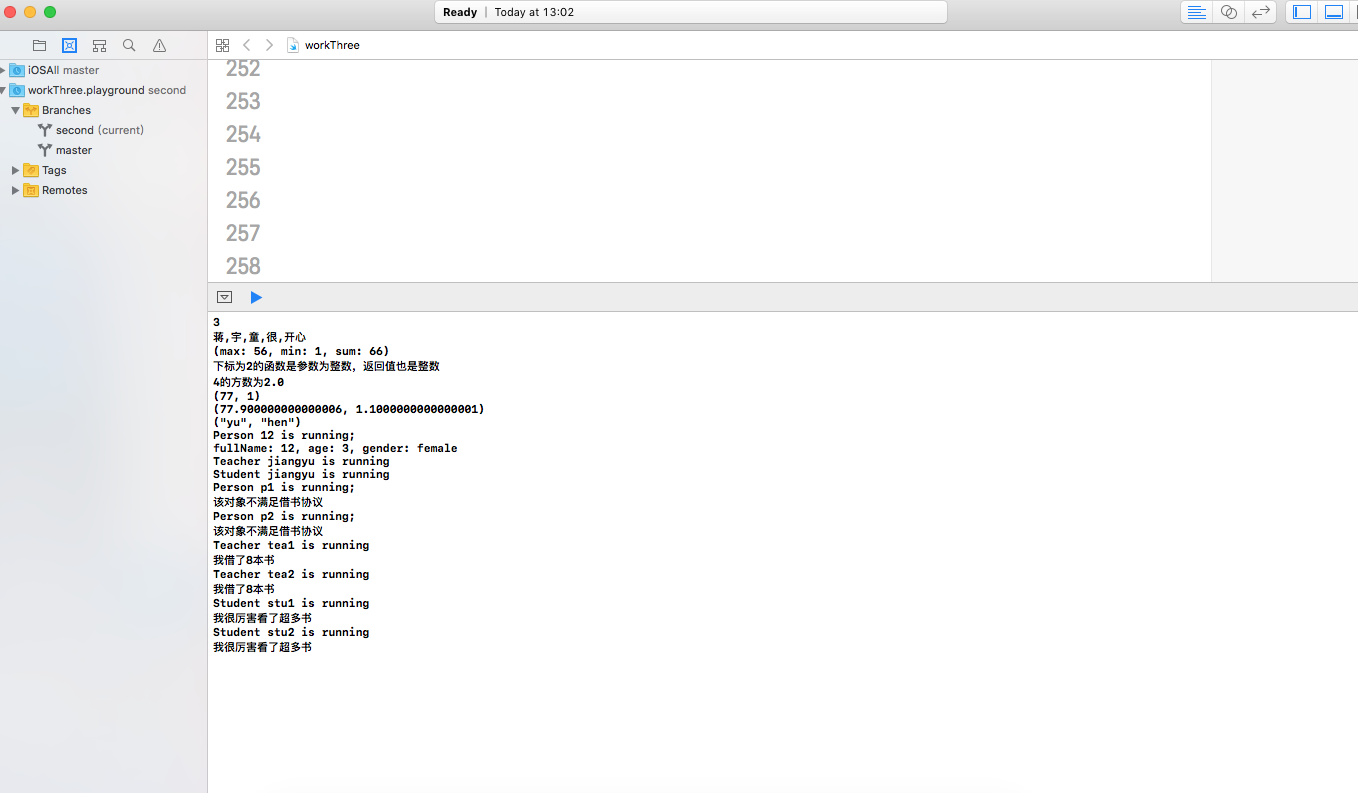
}

* 运行结果：



1. 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；
2. 修改Teacher和Student，让这两个类实现该协议；
3. 对数组执行以下要求：
   1. 分别统计Person、Teacher和Student对象的个数并放入一字典中，统计完后输出字典内容；
   2. 对数组按以下要求排序并输出：age、fullName、gender+age；
   3. 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；

* 程序代码：
* import UIKit
* //（1） 给定一个Dictionary，Dictionary包含key值name和age，用map函数返回age字符串数组;
* var dirctionary:[[String:Any]]=[["name":"蒋宇童","age":20],["name":"蒋开心","age":30],["name":"蒋快乐","age":40]]
* var arr=dirctionary.map{$0["age"]}
* //（2） 给定一个String数组，用filter函数选出能被转成Int的字符串
* var filterInt=["1","2","3","开心","每一天"]
* var arr2=filterInt.filter{Int($0) != nil}
* print(arr2.count)
* //（3） 用reduce函数把String数组中元素连接成一个字符串，以逗号分隔
* var arr3=["蒋","宇","童","很","开心"]
* var co=arr3.count
* var n=0
* var arr4=arr3.reduce("") {
* if n<co-1
* {
* n=n+1
* return $0+$1+","
* }
* else
* {
* n=n+1
* return $0+$1
* }
* }
* print(arr4)
* var arr5=[1,2,3,4,5,6,7,8]
* var arr6=[1,2,3,4,56,]
* let tuple = arr6.reduce((max: arr6[0], min:arr6[0], sum: 0)){ (max($0.max, $1),min($0.min, $1), $0.sum + $1) }
* print(tuple)
* // 新建一个函数数组，函数数组里面保存了不同函数类型的函数，要求从数组里找出参数为一个整数，返回值为一个整数的所有函数；
* func one()->Void
* {
* print("返回值为空")
* }
* func two()->Int
* {
* return 2
* }
* func three(max:Int)->Int
* {
* return 3
* }
* func four(max:Int)->Void
* {
* print("返回值为空，参为整数")
* }
* var arr7:[Any]=[one,two,three,four]
* for (key,value) in arr7.enumerated()
* {
* if value is (Int)->Int
* {
* print("下标为\(key)的函数是参数为整数，返回值也是整数")
* }
* }
* //扩展Int，增加sqrt方法，可以计算Int的Sqrt值并返回浮点数，进行验证；
* extension Int {
* func Sqrt() -> Double {
* return sqrt(Double(self))
* }
* }
* var m=4.Sqrt()
* print("4的方数为\(m)")
* //实现一个支持泛型的函数，该函数接受任意个变量并返回最大和最小值，分别传入整数值、浮点数值、字符串进行验证。
* func found<T:Comparable>(a:T...)->(T,T)
* {
* return a.reduce((max:a[0],min:a[0]), {(max($0.max,$1),min($0.min,$1))})
* }
* print(found(a:1,2,3,4,5,6,77))
* print(found(a:1.1,2,3,4,5,6,77.9))
* print(found(a:"jiang","yu","tong","hen","kai","xing"))
* /\*
* （1） 实现Person类：
* a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
* b) 具有指定构造函数和便利构造函数；
* c) 两个Person实例对象可以用==和!=进行比较；
* d) Person实例可以直接用print输出；
* e) Person增加run方法(方法里面直接print输出Person XXX is running;
* \*/
* enum Gender
* {
* case male
* case female
* }
* class Person:CustomStringConvertible
* {
* var firstName:String
* var lastName:String
* var age:Int
* var gender:Gender
* init(firstName:String, lastName:String,age:Int,gender:Gender) {
* self.firstName=firstName
* self.lastName=lastName
* self.age=age
* self.gender=gender
* }
* var fullName:String
* {
* get
* {
* return firstName+lastName
* }
* }
* convenience init(name:String,age:Int)
* {
* self.init(firstName:name,lastName:"",age:age,gender:Gender.female)
* }
* static func ==(p1:Person,p2:Person)->Bool
* {
* return p1.fullName==p2.fullName&&p1.age==p2.age
* }
* static func != (p1:Person,p2:Person)->Bool
* {
* return p1.fullName != p2.fullName||p1.age != p2.age
* }
* var description: String {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"
* }
* func run ()->Void
* {
* print("Person \(fullName) is running;")
* }
* }
* var per1=Person(firstName: "1", lastName: "2", age: 3, gender: .female)
* per1.run()
* print(per1)
* /\*
* （2） 从Person分别派生Teacher类和Student类：
* a) Teacher类增加属性title，实例可以直接用print输出；
* b) Student类增加属性stuNo，实例可以直接用print输出；
* c) Teacher和Student重载run方法(方法里面直接print输出Teacher XXX is running和Student XXX is running)
* （5） 修改Teacher和Student，让这两个类实现该协议；
* \*/
* /\*
* （4） 新建一个协议SchoolProtocol，协议包括一个department属性(Enum，自己实现enum的定义)和lendBook方法（随便写点内容，能区隔即可）；
* \*/
* enum ProtocolEnum
* {
* case happy
* case lucky
* }
* protocol SelfWrite
* {
* var dapartment:ProtocolEnum{get}//自读
* func lendBook()
* }
* //老师
* class Teacher:Person,SelfWrite
* {
* var dapartment: ProtocolEnum
* {
* return .happy
* }
* func lendBook() {
* print("我借了8本书")
* }
* var title:String
* init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {
* self.title=title
* super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)
* }
* override var description: String
* {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"
* }
* override func run() {
* print("Teacher \(fullName) is running")
* }
* }
* var t1=Teacher(firstName: "jiang", lastName: "yu", age: 20, gender: .female, title: "good")
* t1.run()
* //学生
* class Student:Person,SelfWrite
* {
* var dapartment: ProtocolEnum
* {
* return .lucky
* }
* func lendBook() {
* print("我很厉害看了超多书")
* }
* var stuNo:String
* init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {
* self.stuNo=stuNo
* super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)
* }
* override var description: String
* {
* return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"
* }
* override func run() {
* print("Student \(fullName) is running")
* }
* }
* var s1=Student(firstName: "jiang", lastName: "yu", age: 20, gender: .female, stuNo: "good")
* s1.run()
* /\*
* c) 对数组进行穷举，调用每个对象的run方法，同时调用满足协议SchoolProtocol对象的lendBook方法；
* \*/
* var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)
* var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)
* var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")
* var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")
* var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")
* var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")
* var arry=[p1,p2,tea1,tea2,stu1,stu2]
* for i in arry
* {
* i.run()
* if let a = i as? Student
* {
* a.lendBook()
* }
* else if let a=i as? Teacher
* {
* a.lendBook()
* }
* else
* {
* print("该对象不满足借书协议")
* }
* }
* 运行结果：



1. 实验结果的分析与评价（该部分如不够填写，请另加附页）

**Github：https://github.com/jiangyutong/swiftWork/tree/master/代码**

这次的实验主要是讲的是范型，扩展。继承很有协议这些。我觉得swift中的类的初始化很有意思，它可以有便利构造器，虽然这个是基于指定构造器的，但是给我们实列化对象带来列很大的便利。并且swift它主要是用协议，所以说它给人的感觉就是很轻量级的。

注：实验成绩等级分为（90－100分）优，（80－89分）良，(70-79分)中，（60－69分）及格，（59分）不及格。