Comenzado el	miércoles, 24 de mayo de 2023, 20:02
Estado	Finalizado
Finalizado en	miércoles, 24 de mayo de 2023, 20:03
Tiempo empleado	41 segundos

Pregunta 1

Finalizado

Puntúa como 1,00

Copia a continuación la solución del ejercicio 1 de la práctica

Aaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```
// Solución práctica 12: Programación Orientada a Objetos en Swift (2)

import Foundation


/*
===========
Ejercicio 1
===========
*/


print("""
---
Ejercicio 1
---\n
""")

// a)

protocol A {
    var a: Int {get set}
    func foo(a: Int, b: Int) -> Int?
}

protocol B {
    mutating func bar()
}

// MiStruct debe ajustarse a los protocolos A y B:
//  - debe tener una variable escribible de tipo Int llamada a
//  - debe tener una función foo que reciba dos enteros y devuelva un opcional entero.
//    En nuestro ejemplo hemos definido una función que comprueba la suma de a y b y si es
//    mayor que la propiedad a devuelve la suma, en caso contrario devuelve nil.
//  - debe definir una función mutadora llamada bar. En nuestro ejemplo simplemente incrementa
//    la propiedad a en 1.

struct MiStruct: A, B {
    var a = 10
    func foo(a: Int, b: Int) -> Int? {
        let res = a + b > self.a ? a + b : nil
        return res
    }
    mutating func bar() {
        a += 1
    }
}

// b)
// El error está en la definición de la propiedad `a` como `let` en la estructura `MiStruct`.
// Para que la estructura cumpla el protocolo `A` la propiedad `a` debe ser escribible.


// c)

struct Equipo: Equatable, Comparable {
    let puntos: Int
    let nombre: String

    static func < (primero: Equipo, segundo: Equipo) -> Bool {
        return
            primero.puntos < segundo.puntos ||
            (primero.puntos == segundo.puntos && primero.nombre < segundo.nombre)
    }
}

let equipo1 = Equipo(puntos: 10, nombre: "Hércules")
let equipo2 = Equipo(puntos: 8, nombre: "Villareal")
print(equipo1 == equipo2) // imprime false
print(equipo1 > equipo2) // imprime true

let ejemplo = [
    Equipo(puntos: 5, nombre: "Hercules"),
```

```
    Equipo(puntos: 10, nombre: "Valencia"),
    Equipo(puntos: 8, nombre: "Betis"),
    Equipo(puntos: 1, nombre: "Elche"),
    Equipo(puntos: 5, nombre: "Leganés"),
    Equipo(puntos: 10, nombre: "Barça")
    ]

for e in ejemplo.sorted() {
    print(e)
}
```

Pregunta 2

Finalizado

Puntúa como 1,00

Copia a continuación la solución del ejercicio 2 de la práctica

Aaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```
/*
===========
Ejercicio 2
===========
*/


print("""
\n---
Ejercicio 2
---\n
""")


// a)

struct Cuadrado {
    var lado: Double
}

extension Cuadrado {
    var area: Double {
        return lado * lado
    }
}

var cuadrado = Cuadrado(lado: 4.0)
print("Área del cuadradado: \(cuadrado.area)") // Imprime: 16.0
cuadrado.lado = 10.0
print("Área del cuadrado: \(cuadrado.area)") // Imprime: 100.0


// b)

protocol Persona {
    var nombre: String {get}
    func encantada() -> Persona
    func refrescada() -> Persona
}

enum Pocion {
    case magica, refrescante, venenosa

    func esBebida(por persona: Persona) -> Persona? {
        switch self {
            case .magica:
                return persona.encantada()
            case .refrescante:
                return persona.refrescada()
            default:
                return nil
        }
    }
}


// c)

protocol Ac {
    var valor: Int {get set}
    func foo(a: Int) -> Int
}
protocol Bc {
    mutating func bar()
}
struct MiStruct2: Ac, Bc {
    var valor = 10
    func foo(a: Int) -> Int {
        return a + valor
    }
    mutating func bar() {
        valor += 1
    }
```

```
}

// d)

struct CirculoV2 {
    var radio: Double
    static func + (primero: CirculoV2, segundo: CirculoV2) -> CirculoV2 {
        return CirculoV2(radio: primero.radio + segundo.radio)
    }
}

let c1 = CirculoV2(radio: 5.0)
let c2 = CirculoV2(radio: 10.0)
let c3 = c1 + c2
print("El radio de la suma es: \(c3.radio)")
// Imprime: El radio de la suma es: 15.0
```

Pregunta 3

Finalizado

Puntúa como 1,00

Copia a continuación la solución del ejercicio 3 de la práctica

Aaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```
/*
===========
Ejercicio 3
===========
*/


print("""
\n---
Ejercicio 3
---\n
""")

// a)

protocol P {
   var p: Int { get }
}
class A1: P {
   var p = 0
   var a1 = 0
}
class A2: P {
   var p = 1
   var a2 = 0
}

var array: [P] = [A1(), A2()]
for i in array {
   if let x = i as? A1 {
       print("p: \(i.p), a1: \(x.a1)")
   } else if let x = i as? A2 {
       print("p: \(i.p), a2: \(x.a2)")
   }
}

// debe imprimir:
// p: 0, a1: 0
// p: 1, a2: 0

// b)

protocol TieneVelocidad {
    func velocidadActual () -> Double
}

class Vehiculo {
    var velocidad = 0.0
    func velocidadActual() -> Double {
        return velocidad
    }
}

class Tren {
    static let velocidadEnMarcha = 300.0
    var pasajeros = 0
    var enMarcha = false
}

extension Vehiculo: TieneVelocidad {}
extension Tren: TieneVelocidad {
    func velocidadActual() -> Double {
        return enMarcha ? Tren.velocidadEnMarcha : 0.0
    }
}

var vehiculo1 = Vehiculo()
var tren1 = Tren()
tren1.enMarcha = true

let transportes: [TieneVelocidad] = [vehiculo1, tren1]

for i in transportes {
```

```
    print(i.velocidadActual())
}
```

Pregunta 4

Finalizado

Puntúa como 1,00

Copia a continuación la solución del ejercicio 4 de la práctica

Aaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```
/*
==========
Ejercicio 4
==========
*/


print("""
\n---
Ejercicio 4
---\n
""")

struct Timer {
    var segundos: Int
    mutating func paso() {
        if (segundos > 0) {
            segundos -= 1
            Timer.pasosTotales += 1
            if (segundos == 0) {
                print("Tiempo!!!!")
            }
        }
    }
    static func + (primero: Timer, segundo: Timer) -> Timer {
        return Timer(segundos: primero.segundos + segundo.segundos)
    }
    static var pasosTotales = 0
}

var t1 = Timer(segundos: 10)
var t2 = Timer(segundos: 5)
for _ in 0...4 {
    t1.paso()
}
for _ in 0...2 {
    t2.paso()
}
var t3 = t1 + t2
t3.paso()
print("Segundos del temporizador 1: \(t1.segundos)")
print("Segundos del temporizador 2: \(t2.segundos)")
print("Segundos del temporizador 3: \(t3.segundos)")
print("Pasos totales: \(Timer.pasosTotales)")
// Imprime:
// Segundos del temporizador 1: 5
// Segundos del temporizador 2: 2
// Segundos del temporizador 3: 6
// Pasos totales: 9
```

Pregunta 5

Finalizado

Puntúa como 1,00

Copia a continuación la solución del ejercicio 5 de la práctica

Aaasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```
/*
==========
Ejercicio 5
==========
*/


print("""
\n---
Ejercicio 5
---\n
""")

struct Punto {
    var x = 0.0, y = 0.0
}

struct Tamaño {
    var ancho = 0.0, alto = 0.0
}

protocol Figura {
    var centro: Punto {get set}
    var area: Double {get}
    var tamaño: Tamaño {get}
    func descripcion() -> String
}

struct Circulo: Figura {
    var centro = Punto()
    var radio = 0.0

    var area: Double {
        get {
            return Double.pi * radio * radio
        }
        set {
            radio = sqrt(newValue / Double.pi)
        }
    }
    var tamaño: Tamaño {
        let diametro = radio * 2
        return Tamaño(ancho: diametro, alto: diametro)
    }

    func descripcion() -> String {
        return "Centro: \(centro) y área: \(area)"
    }
}

struct Rectangulo: Figura {
    var origen = Punto()
    var tamaño = Tamaño()

    var centro: Punto {
        get {
            let centroX = origen.x + (tamaño.ancho / 2)
            let centroY = origen.y + (tamaño.alto / 2)
            return Punto(x: centroX, y: centroY)
        }
        set {
            origen.x = newValue.x - (tamaño.ancho / 2)
            origen.y = newValue.y - (tamaño.alto / 2)
        }
    }

    var area: Double {
        return tamaño.ancho * tamaño.alto
    }

    func descripcion() -> String {
        return "Centro: \(centro) y área: \(area)"
    }
```

```
}

struct AlmacenFiguras {
    var figuras: [Figura] = []

    mutating func añade(figura: Figura) {
        figuras.append(figura)
    }

    var numFiguras: Int {
        return figuras.count
    }

    var areaTotal: Double {
        func areas() -> [Double] {
            return
                figuras.map() {$0.area}
        }
        return areas().reduce(0.0, +)
    }

    func cuentaTipos() -> (Int, Int) {
        var nRect = 0, nCirc = 0
        for fig in figuras {
            switch fig {
                case let rect as Rectangulo:
                    nRect += 1
                    print("** Un rectángulo con tamaño \(rect.tamaño) y descripción:")
                case is Circulo:
                    nCirc += 1
                    print("** Un círculo con descripción:")
                default:
                    print("Tipo de figura desconocida")
            }
            print(fig.descripcion())
        }
        return (nRect, nCirc)
    }
}


let r = Rectangulo(origen:Punto(x:3, y:4), tamaño:Tamaño(ancho:10, alto:5))
let c = Circulo(centro:Punto(x:5, y:0), radio:10.0)
var almacen = AlmacenFiguras()
almacen.añade(figura: r)
almacen.añade(figura: c)
print("Total figuras: \(almacen.cuentaTipos())")
```