Asignatura: Introducción al desarrollo con Swift

Guía de Ejercicios N.º 3

6 de Julio de 2024

Managua, 29 de Junio de 2024

INTRODUCCIÓN AL DESARROLLO CON SWIFT

I. CONTENIDOS

Ejercicio 1: Enumeraciones y Delegación

Implementar una enumeración `EstadoPedido` con casos `procesando`, `enviado`, y `entregado`. Crear un protocolo `NotificacionPedido` con un método `notificarCambioDeEstado`. Implementar una clase `Pedido` que tenga una propiedad `estado` de tipo `EstadoPedido` y una propiedad `delegado` de tipo `NotificacionPedido`. Crear una función que cambie el estado del pedido y notifique al delegado.

```
ENUM EstadoPedido
    CASO procesando
    CASO enviado
    CASO entregado
END ENUM

PROTOCOL NotificacionPedido
    FUNC notificarCambioDeEstado(nuevoEstado: EstadoPedido) -> VOID
END PROTOCOL

CLASE Pedido
    VAR estado: EstadoPedido
    VAR delegado: NotificacionPedido?

    FUNC cambiarEstado(nuevoEstado: EstadoPedido)
        estado = nuevoEstado
        SI delegado != NULL ENTONCES
            delegado.notificarCambioDeEstado(nuevoEstado)
        FIN SI
    END FUNC
END CLASE

CLASE GestorPedido IMPLEMENTS NotificacionPedido
    FUNC notificarCambioDeEstado(nuevoEstado: EstadoPedido)
        IMPRIMIR("El estado del pedido ha cambiado a: " + nuevoEstado)
    END FUNC
END CLASE

// Ejemplo de uso
VAR gestor = GestorPedido()
VAR pedido = Pedido(estado: EstadoPedido.procesando)
pedido.delegado = gestor
pedido.cambiarEstado(EstadoPedido.enviado)
```

Ejercicio 2: Funciones Estáticas y Clases

Crear una clase `Calculadora` con funciones estáticas `sumar` y `restar`. Implementar una función que acepte un array de enteros y use las funciones estáticas para calcular la suma y la resta total de los elementos del array.

```
CLASE Calculadora
    STATIC FUNC sumar(a: INT, b: INT) -> INT
        RETURN a + b
    END FUNC

    STATIC FUNC restar(a: INT, b: INT) -> INT
        RETURN a - b
    END FUNC
END CLASE

FUNC calcularSumaYRestaTotales(numeros: [INT]) -> (sumaTotal: INT, restaTotal: INT
    VAR sumaTotal = 0
    VAR restaTotal = 0

    PARA cada numero EN numeros
        sumaTotal = Calculadora.sumar(sumaTotal, numero)
        restaTotal = Calculadora.restar(restaTotal, numero)
    FIN PARA

    RETURN (sumaTotal, restaTotal)
END FUNC

// Ejemplo de uso
VAR numeros = [1, 2, 3, 4, 5]
VAR resultado = calcularSumaYRestaTotales(numeros)
IMPRIMIR("Suma total: " + resultado.sumaTotal)
IMPRIMIR("Resta total: " + resultado.restaTotal)
```

Ejercicio 3: Polimorfismo y Structs

Descripción: Crear un protocolo `Figura` con un método `calcularArea`. Implementar estructuras `Cuadrado` y `Rectangulo` que conformen al protocolo `Figura`. Crear una función que acepte un array de `Figura` y calcule el área total de todas las figuras.

```
PROTOCOL Figura
    FUNC calcularArea() -> DOUBLE
END PROTOCOL

STRUCT Cuadrado IMPLEMENTS Figura
    VAR lado: DOUBLE

    FUNC calcularArea() -> DOUBLE
        RETURN lado * lado
    END FUNC
END STRUCT

STRUCT Rectangulo IMPLEMENTS Figura
    VAR ancho: DOUBLE
    VAR alto: DOUBLE

    FUNC calcularArea() -> DOUBLE
        RETURN ancho * alto
    END FUNC
END STRUCT
```

```
FUNC calcularAreaTotal(figuras: [Figura]) -> DOUBLE
    VAR areaTotal = 0.0

    PARA cada figura EN figuras
        areaTotal = areaTotal + figura.calcularArea()
    FIN PARA

    RETURN areaTotal
END FUNC

// Ejemplo de uso
VAR cuadrado = Cuadrado(lado: 4.0)
VAR rectangulo = Rectangulo(ancho: 3.0, alto: 5.0)
VAR figuras: [Figura] = [cuadrado, rectangulo]

VAR areaTotal = calcularAreaTotal(figuras)
IMPRIMIR("Área total: " + areaTotal)
```

Ejercicio 4: Enums y Funciones Estáticas

Implementar una enumeración `OperacionMatematica` con casos `suma`, `resta`, `multiplicacion`, y `division`. Crear una clase `OperadorMatematico` con una función estática `realizarOperacion` que acepte dos números y un caso de `OperacionMatematica`, y devuelva el resultado de la operación.

```
ENUM OperacionMatematica
    CASO suma
    CASO resta
    CASO multiplicacion
    CASO division
END ENUM

CLASE OperadorMatematico
    STATIC FUNC realizarOperacion(a: DOUBLE, b: DOUBLE, operacion: OperacionMatemat
        SWITCH operacion
            CASO suma
                RETURN a + b
            CASO resta
                RETURN a - b
            CASO multiplicacion
                RETURN a * b
            CASO division
                SI b != 0
                    RETURN a / b
                SINO
                    IMPRIMIR("Error: División por cero")
                    RETURN 0
                FIN SI
        END SWITCH
    END FUNC
END CLASE

// Ejemplo de uso
VAR resultadoSuma = OperadorMatematico.realizarOperacion(10, 5, OperacionMatematica
IMPRIMIR("Resultado de la suma: " + resultadoSuma)

VAR resultadoDivision = OperadorMatematico.realizarOperacion(10, 0, OperacionMatema
IMPRIMIR("Resultado de la división: " + resultadoDivision)
```

Ejercicio 5: Clases, Structs y Polimorfismo

Crear un protocolo `Empleado` con un método `calcularSalario`. Implementar una clase `EmpleadoBase` y una estructura `EmpleadoContrato` que conformen al protocolo `Empleado`. Crear una función que acepte un array de `Empleado` y calcule el salario total de todos los empleados.

```
PROTOCOL Empleado
    FUNC calcularSalario() -> DOUBLE
END PROTOCOL

CLASE EmpleadoBase IMPLEMENTS Empleado
    VAR salarioBase: DOUBLE

    INIT(salarioBase: DOUBLE)
        SELF.salarioBase = salarioBase
    END INIT

    FUNC calcularSalario() -> DOUBLE
        RETURN salarioBase
    END FUNC
END CLASE

STRUCT EmpleadoContrato IMPLEMENTS Empleado
    VAR tarifaHora: DOUBLE
    VAR horasTrabajadas: DOUBLE

    FUNC calcularSalario() -> DOUBLE
        RETURN tarifaHora * horasTrabajadas
    END FUNC
END STRUCT
```

```
FUNC calcularSalarioTotal(empleados: [Empleado]) -> DOUBLE
    VAR salarioTotal = 0.0

    PARA cada empleado EN empleados
        salarioTotal = salarioTotal + empleado.calcularSalario()
    FIN PARA

    RETURN salarioTotal
END FUNC

// Ejemplo de uso
VAR empleadoBase = EmpleadoBase(salarioBase: 3000.0)
VAR empleadoContrato = EmpleadoContrato(tarifaHora: 20.0, horasTrabajadas: 100.0)
VAR empleados: [Empleado] = [empleadoBase, empleadoContrato]

VAR salarioTotal = calcularSalarioTotal(empleados)
IMPRIMIR("Salario total: " + salarioTotal)
```