



```
PLANET_POS = {x = 0, y = 0}
```

```
ANGLE = 0
```

```
DISTANCE = 30
```

```
-- UPDATE FUNCTION
```

```
FUNCTION UPDATE()
```

```
    ANGLE = ANGLE + 0.01
```

```
    PLANET_POS.x = COS(ANGLE) * DISTANCE
```

```
    PLANET_POS.y = SIN(ANGLE) * DISTANCE
```

```
END
```

```
-- DRAW FUNCTION
```

```
FUNCTION DRAW()
```

```
CLEAR()
```

```
CIRCLE(50 + PLANET_POS.x, 50 + PLANET_POS.y, 5)
```

```
END
```

PRESENTACIÓN

LUA

BY JONATHAN MUÑOZ MORALES



INTRODUCCIÓN

Todos los ejemplos serán en Lua.

Es un lenguaje de script embebido, lo que lo hace ligero y eficiente.

Es un lenguaje que soporta programación orientada a objetos, funcional, gestión de datos, descripción de datos.

Temas:

- Encapsulación
- Inyección
- Lambdas



ENCAPSULACIÓN EJEMPLO 1

ENCAPSULACIÓN

- Ocultación de la información
- Protección de datos
- Modularidad y reutilización
- Abstracción
- Mantenimiento y evolución
- Organización y estructura

```
class Coche {
```

```
// Atributos
```

```
public int número_de_ruedas;  
public int litros_gasolina;  
public String modelo;
```

```
// Funciones o métodos
```

```
public void arrancar();  
public void acelerar();  
public void frenar();
```

```
}
```

ENCAPSULACIÓN

local Account = {}



ENCAPSULACIÓN

```
local Account = {}
```

```
function Account:new(o)
```



ENCAPSULACIÓN

```
local Account = {}
```

```
function Account:new(o)
```

```
o = o or {}
```



ENCAPSULACIÓN

```
local Account = {}
```

```
function Account:new(o)
    o = o or {}
    setmetatable(o, self)
```



ENCAPSULACIÓN

```
local Account = {}  
  
function Account:new(o)  
    o = o or {}  
    setmetatable(o, self)  
    self.__index = self
```

ENCAPSULACIÓN

```
local Account = {}  
  
function Account:new(o)  
    o = o or {}  
    setmetatable(o, self)  
    self.__index = self  
    return o  
end
```



ENCAPSULACIÓN

```
local Account = {}  
  
function Account:new(o)  
    o = o or {}  
    setmetatable(o, self)  
    self.__index = self  
    return o  
end
```

```
s = Account:new{cantidad = 0.00}--damos de alta una cuenta c
```

ENCAPSULACIÓN

```
function Account:deposit(c)
    self.cantidad = self.cantidad + c
end

function Account:withdraw(c)
    if c > self.cantidad then
        error("saldo insuficiente")
    end
    self.cantidad = self.cantidad - c
end
```

```
local Account_mt = {
    __index = function(t, k)
        if k == "cantidad" then
            return t._o
        elseif k == "deposit" then
            return function(self, c)
                self:_deposit(c)
            end
        elseif k == "withdraw" then
            return function(self, c)
                self:_withdraw(c)
            end
        else
            error("intentando acceder a un elemento no declarado_ 0" .. k)
        end
    end,
    __newindex = function(t, k, v)
        error("intentando acceder a un elemento no declarado: " .. k)
    end,
}

function Account:_deposit(c)
    self.cantidad = self._cantidad + c
end

function Account:_withdraw(c)
    if c > self._cantidad then
        error("saldo insuficiente")
    end
    self._cantidad = self._cantidad - c
end

setmetatable(Account, Account_mt)
```



INYECCIÓN EJEMPLO 2

INYECCIÓN

- Disminuimos el desacoplamiento
- Aumentamos la reutilización
- Testabilidad
- Flexibilidad

INYECCIÓN

- Disminuimos el desacoplamiento
- Aumentamos la reutilización
- Testabilidad
- Flexibilidad

```
local Motor = {}

function Motor:new()
    local o = {}
    self.__index = self
    setmetatable(o, self)
    o.arrancar = function() self.turnOn = 1 self.marchas = 0 end
    o.apagar = function() self.turnOn = 0 end
    o.aumentarMarcha = function() self.marchas = self.marcha + 1 end
    o.disminuirMarcha = function() self.marchas = self.marcha - 1 end
    o.puntoMuerto = function() self.marchas = -1 end
    o.encendido = function() return self.turnOn end
    o.marchasMotor = function() return self.marchas end
    return o
end

--Utilizando las funciones set... permitimos que los usuario
function Motor:setArrancar(arrancarFunc)
    self.arrancar = arrancarFunc
end

function Motor:setApagar(apagarFunc)
    self.apagar = apagarFunc
end

function Motor:setAumentarMarcha(aumentarMarchaFunc)
    self.aumentarMarcha = aumentarMarchaFunc
end

function Motor:setDisminuirMarcha(disminuirMarchaFunc)
    self.disminuirMarcha = disminuirMarchaFunc
end

function Motor:setPuntoMuerto(puntoMuertoFunc)
    self.puntoMuerto = puntoMuertoFunc
end

return Motor
```

```

function test_Encender()

    print("Encendemos el motor...")
    m:arrancar()
    lunatest.assert_equal(1, m:encendido())

    print("Aumentamos una marcha...")
    m:aumentarMarcha()
    lunatest.assert_equal(1, m:marchasMotor())

    print("Aumentamos una marcha mas...")
    m:aumentarMarcha()
    lunatest.assert_equal(2, m:marchasMotor())

    print("Disminuimos una marchas debido al tráfico...")
    m:disminuirMarcha()
    lunatest.assert_equal(1, m:marchasMotor())

    print("Encontramos aparcamiento, por lo que ponemos punto muerto")
    m:puntoMuerto()
    lunatest.assert_equal(-1, m:marchasMotor())

    print("Apagamos el motor...")
    m:apagar()
    lunatest.assert_equal(0, m:encendido())

    print("Probamos a cambiar la polaridad de arrancar y encender")
    local customArrancar = function(x) Motor.turnOn = 0 end
    m:setArrancar(customArrancar)
    m:arrancar()
    lunatest.assert_equal(0, m:encendido())

    local customApagar = function(x) Motor.turnOn = 1 end
    m:setApagar(customApagar)
    m:apagar()
    lunatest.assert_equal(1, m:encendido())

end

```



```

local Motor = {}

function Motor:new()
    local o = {}
    self.__index = self
    setmetatable(o, self)
    o.arrancar = function() self.turnOn = 1 self.marchas = 0 end
    o.apagar = function() self.turnOn = 0 end
    o.aumentarMarcha = function() self.marchas = self.marcha + 1 end
    o.disminuirMarcha = function() self.marchas = self.marcha - 1 end
    o.puntoMuerto = function() self.marchas = -1 end
    o.encendido = function() return self.turnOn end
    o.marchasMotor = function() return self.marchas end
    return o
end

--Utilizando las funciones set... permitimos que los usuarios
function Motor:setArrancar(arrancarFunc)
    self.arrancar = arrancarFunc
end

function Motor:setApagar(apagarFunc)
    self.apagar = apagarFunc
end

function Motor:setAumentarMarcha(aumentarMarchaFunc)
    self.aumentarMarcha = aumentarMarchaFunc
end

function Motor:setDisminuirMarcha(disminuirMarchaFunc)
    self.disminuirMarcha = disminuirMarchaFunc
end

function Motor:setPuntoMuerto(puntoMuertoFunc)
    self.puntoMuerto = puntoMuertoFunc
end

return Motor

```



LAMBDA
EJEMPLO 3

LAMBDA

- Reducimos código repetitivo
- Concisión y legibilidad
- Flexibilidad y expresividad

LAMBDA

```
function(element) return element % n == 0 end)
```

- Reducimos código repetitivo
- Concisión y legibilidad
- Flexibilidad y expresividad

LAMBDA

- Reducimos código repetitivo
- Concisión y legibilidad
- Flexibilidad y expresividad

```
function Lista:mostrarDivisores(n)
    return function()
        local listaPar = Lista:filtro(self.list,
            function(element) return element % n == 0 end)
        if listaPar ~= nil then
            io.write("Los valores pares de la lista
son: ( ")
            for _, element in pairs(listaPar) do
                io.write(element .. " ")
            end
            print(")")
        else
            print("La lista esta vacia...")
        end
    end
end
```

LAMBDA

```
function Lista:filtro(tbl, predicate)
local result = {}
for i, v in ipairs(tbl) do
    if predicate(v) then
        table.insert(result, v)
    end
end
return result
end
```

```
function Lista:mostrarDivisores(n)
return function()
local listaPar = Lista:filtro(self.list,
function(element) return element % n == 0 end)
if listaPar ~= nil then
    io.write("Los valores pares de la lista
son: ( ")
for _, element in pairs(listaPar) do
    io.write(element .. " ")
end
print(")")
else
    print("La lista esta vacia...")
end
end
end
```



GRACIAS