Swift

Ciclo de vida de las instancias

Inicialización

Inicialización

- Es el proceso que prepara una instancia para su uso
- Hasta que no se completa, no queda fijado el estado de la instancia
- Se realiza en métodos especiales llamados inicializadores o mediante la asignación de un valor predeterminado
- Los inicializadores no devuelven valor
- Las clases y estructuras deben definir un valor para todas sus propiedades almacenadas en la inicialización

Inicializadores

```
init() {
    // perform some initialization here
}
```

Inicializadores

```
struct Fahrenheit {
    var temperature: Double
    init() {
        temperature = 32.0
    }
}

var f = Fahrenheit()

print("The default temperature is \(f.temperature)° Fahrenheit")
// Prints "The default temperature is 32.0° Fahrenheit"
```

Asignación de valores predeterminados

```
struct Fahrenheit {
    var temperature = 32.0
}
```

Inicializador con parámetros

```
struct Celsius {
    var temperatureInCelsius: Double
    init(fromFahrenheit fahrenheit: Double) {
        temperatureInCelsius = (fahrenheit - 32.0) / 1.8
    }
    init(fromKelvin kelvin: Double) {
        temperatureInCelsius = kelvin - 273.15
    }
}

let boilingPointOfWater = Celsius(fromFahrenheit: 212.0)
// boilingPointOfWater.temperatureInCelsius is 100.0

let freezingPointOfWater = Celsius(fromKelvin: 273.15)
// freezingPointOfWater.temperatureInCelsius is 0.0
```

Etiquetas de argumentos

- Como todos los inicializadores se llaman `init`, Swift crea etiquetas de argumentos para cada parámetro que pongamos en el inicializador para distinguirlos
- Podemos evitarlo poniendo _ como nombre de argumento
- Al llamar al inicializador siempre hay que poner los nombres de los argumentos que estén definidos

Etiquetas de argumentos

```
struct Color {
    let red, green, blue: Double
    init(red: Double, green: Double, blue: Double) {
        self.red   = red
        self.green = green
        self.blue  = blue
    }
    init(white: Double) {
        red   = white
        green = white
        blue  = white
    }
}
```

Etiquetas de argumentos

```
let magenta = Color(red: 1.0, green: 0.0, blue: 1.0)
let halfGray = Color(white: 0.5)

let veryGreen = Color(0.0, 1.0, 0.0)
// this reports a compile-time error - argument labels are required
```

Parámetros sin etiquetas de argumentos

```
struct Celsius {
    var temperatureInCelsius: Double
    init(fromFahrenheit fahrenheit: Double) {
        temperatureInCelsius = (fahrenheit - 32.0) / 1.8
    }
    init(fromKelvin kelvin: Double) {
        temperatureInCelsius = kelvin - 273.15
    }
    init(_ celsius: Double) {
        temperatureInCelsius = celsius
    }
}

let bodyTemperature = Celsius(37.0)
// bodyTemperature.temperatureInCelsius is 37.0
```

Características de los inicializadores

- Las propiedades opcionales se inicializan automáticamente a `nil`
- Durante la inicialización podemos modificar las propiedades constantes de la instancia, sólo se fijan cuando termina el `init()`
- Las subclases no pueden modificar las constantes heredadas

Inicialización de opcionales

```
class SurveyQuestion {
    var text: String
    var response: String?
    init(text: String) {
        self.text = text
    }
    func ask() {
        print(text)
    }
}

let cheeseQuestion = SurveyQuestion(text: "Do you like cheese?")
cheeseQuestion.ask()
// Prints "Do you like cheese?"

cheeseQuestion.response = "Yes, I do like cheese."
```

Inicialización de constantes

```
class SurveyQuestion {
    let text: String
    var response: String?
    init(text: String) {
        self.text = text
    }
    func ask() {
        print(text)
    }
}

let beetsQuestion = SurveyQuestion(text: "How about beets?")
beetsQuestion.ask()
// Prints "How about beets?"

beetsQuestion.response = "I also like beets. (But not with cheese.)"
```

Inicializador por defecto

- Swift proporciona este inicializador si no hay ninguno definido y todas las propiedades tienen un valor predeterminado
- Este inicializador no recibe parámetros

Inicializador por defecto

```
class ShoppingListItem {
    var name: String?
    var quantity = 1
    var purchased = false
}

var item = ShoppingListItem()
```

Inicializador miembro a miembro para estructuras

- Se genera si no hay definidos inicializadores propios
- No importa que las propiedades de la estructura no tengan un valor predeterminado

Inicializador miembro a miembro para estructuras

```
struct Size {
    var width = 0.0, height = 0.0
}

let twoByTwo = Size(width: 2.0, height: 2.0)
```

ARC

ARC

- Automatic Reference Counting
- Cuando se generan instancias, ARC reserva automáticamente la memoria
- Si una instancia ya no se necesita, la memoria se libera automáticamente
- Para saber si una instancia no está en uso, se lleva un recuento del número de referencias que la apuntan y cuando llega a 0 se libera
- Mientras haya por lo menos una “referencia fuerte” (una referencia almacenada en una propiedad, constante o variable) la instancia seguirá existiendo

Desinicialización

Desinicialización

- Es el proceso que se ejecuta cuando se liberan los recursos de una instancia mediante ARC
- Sólo está disponible para clases
- Se realiza mediante el método especial `deinit()`
- Normalmente sólo lo implementaremos cuando tengamos que liberar recursos externos (ficheros, conexiones de red...)

Desinicialización

```
deinit {
    // perform the deinitialization
}
```

Desinicialización

```
class Bank {
    static var coinsInBank = 10_000
    static func distribute(coins numberOfCoinsRequested: Int) -> Int {
        let numberOfCoinsToVend = min(numberOfCoinsRequested, coinsInBank)
        coinsInBank -= numberOfCoinsToVend
        return numberOfCoinsToVend
    }
    static func receive(coins: Int) {
        coinsInBank += coins
    }
}
```

Desinicialización

```
class Player {
    var coinsInPurse: Int
    init(coins: Int) {
        coinsInPurse = Bank.distribute(coins: coins)
    }
    func win(coins: Int) {
        coinsInPurse += Bank.distribute(coins: coins)
    }
    deinit {
        Bank.receive(coins: coinsInPurse)
    }
}
```

Desinicialización

```
var playerOne: Player? = Player(coins: 100)

print("A new player has joined the game with \(playerOne!.coinsInPurse) coins")
// Prints "A new player has joined the game with 100 coins"

print("There are now \(Bank.coinsInBank) coins left in the bank")
// Prints "There are now 9900 coins left in the bank"
```

Desinicialización

```
playerOne!.win(coins: 2_000)

print("PlayerOne won 2000 coins & now has \(playerOne!.coinsInPurse) coins")
// Prints "PlayerOne won 2000 coins & now has 2100 coins"

print("The bank now only has \(Bank.coinsInBank) coins left")
// Prints "The bank now only has 7900 coins left"
```

Desinicialización

```
playerOne = nil

print("PlayerOne has left the game")
// Prints "PlayerOne has left the game"

print("The bank now has \(Bank.coinsInBank) coins")
// Prints "The bank now has 10000 coins"
```