Conceptos de Swift 2

PMDM - iOS Programación multimedia y dispositivos móviles

Ariel Hernández, Febrero 2021

Swift: Conceptos

- `enum`
- *Optionals*

Swift: `enum`

- Otro tipo de datos como `struct` y `class`
  - Solo puede tomar valores discretos

```
enum Shape {
  case Triangle
  case Circle
  case Square
  case Line
}
```

  - Es *value type* como los `struct`

- Pueden tener “associated data"

```
enum Shape {
  case Triangle(side1: Int, side2: Int)
  case Square(Int)
  case Circle(radius: Int)
  case Line
}
```

Swift: enum

Como usamos los enum?

```
var circle: Shape = Shape.circle(radius: 3)
let square: Shape = .square(3)
let line = .line //Swift no puede inferir el tipo
```

Swift: `switch`

```
switch shape {
   case .triangle: print("triangle")
   case .square,
        .rectangle: print(“square")
   case .circle: break
   default: print(“other")
}
```

- Si no interesa un `case` usamos `break`
- Un `switch` tiene que manejar TODOS los valores
- Si no queremos cubrir todos los casos usamos `default`
- Si vamos a capturar varios valores en un mismo `case`, los separamos por `“,"`

- Y los *associated values*?

Swift: `switch`

```
switch shape {
    case .triangle(let side1, _):
        print("triangle with first side: \(side1)")
    case .square:
        break
    case .circle(let radius):
        print(“circle with radius: \(radius)")
    default:
        break
}
```

- Las constantes toman el valor y tipo de la definición del valor del `enum`

Swift: enum

```
enum Shape {
    ...
    var isPolygon: Bool {
        switch self {
        case .circle,.line: return false
        default: return true
        }
    }

    func sides() -> Int {
        switch self {
        case .triangle: return 3
        case .square: return 4
        default: 0
        }
    }
}
```

- Puede tener `func`s
- Puede tener `var`s “computadas"
- **NO** puede tener `var`s “almacenadas”

- El estado del enum se compone **únicamente** por su caso y su *associated data*

Swift: Optionals

- Un *Optional* es un `enum`, nada más. (Bueno, es un poco especial pero… no tanto)

```
enum Optional<T> {
    case nil
    case some(T)
}
```

Tiene cosas especiales:

- Se definen de un modo especial
  `var circle: Int?`
  `circle` puede ser `nil` o tener valor de tipo `Int`

- SIEMPRE tiene definición, por defecto puede sería `nil`

Swift: Optionals

- Puede ser “*unwrapped*” para obtener el valor de la variable **a la fuerza** con `!`

```
var card: Card?
print(card!.content)
```

  - Si `card == nil`, la ejecución terminará con una excepción (o un *crash*)

- Se puede “*unwrap*” de una manera segura

```
if let safeCard = card {
    // safeCard no será nunca nil
    print(card.content)
}

let joker = card ?? Card("🏴‍☠️", 5)

guard let safeCard = card else { return “🏴‍☠️”}
// safeCard no será nunca nil
return safeCard.content
```

Demo

- Layout
- Optionals
- `enum`s