UIKit

PMDM - iOS Programación multimedia y dispositivos móviles

Ariel Hernández, Febrero 2021

Swift: Errores

- En Swift los errores se representan con objetos Value Type que definen el protocolo Error `Error`
- Los `enum` son un buen recurso para definir Errores
- “Lanzar” un error se utiliza para informar de la ocurrencia de una situación inesperada.
- Para lanzar un error se usa la instrucción `throw`

```
enum InputError: Error {
    case fileNotExistError
    case readError
    case writeError
}

throw InputError.readError
```

Swift: Errores

- Cuando un error es lanzado, algún código en la ejecución de ese código TIENE que manejar ese error.
- Existen 4 maneras de manejar un error en Swift:
  - Propagar el error a la función que llamó al código que genera el error.
  - Manejar el error usando la instrucción `do-catch`.
  - Manejar el error como un Optional Value.
  - Asegurar que el error no ocurre.

Swift: Errores, `throws`

- Si una función puede lanzar un error, debe mencionarlo en su definición.
- Si una función declara que puede lanzar errores, estos errores tienen que ser manejado por el código que le llama.
- Para lanzar un error se usa la instrucción `throw`
- Para llamar a una función que puede lanzar errores es necesario usar la instrucción `try`

```
func throwsError() throws {}

func propagateError() throws
{
    try throwsError()
}
```

Swift: Errores, do-catch

- Para manejar directamente los errores se utiliza la instrucción `do` que capturará cualquier Error lanzado en su bloque de código.
- Los errores serán capturados por las cláusulas `catch`
- Si el error no corresponde con ninguna cláusula específica, es capturado en el `catch` final, donde es definida una variable `error` con el error capturado.
- Si no se define una cláusula `catch` abierta, entonces los errores no capturados son propagados.

```
func propagateError() throws {
    do {
        try throwsError()
    } catch InputError.readError,
            InputError.writeError{
        print(“I/O Error")
    } catch {
        print("Other Error \(error)")
    }
}
```

Swift: Errores *opcionales*

- Podemos usar `try?` para convertir en optional el resultado de una función que puede lanzar error.
- Si ocurre algún error, entonces el valor de la variable será `nil`
- Manejar los errores como opcionales, permite escribir código conciso e ignorar errores no relevantes.

```
let error = try? throwsError()
```

Swift: Errores no propagados

- A veces sabemos que un método no lanzará error, aunque lo defina. Para esos casos usamos `try!`
- Al igual que ocurre con los Optionals, esta instrucción fallará en ejecución si ocurre algún error.
- Al usar esta instrucción no necesitamos propagar más el error.

```
let noError = try! throwsError()
```

Demo

- `UIKit`
- `NSUrl`
- `JSON`
- Errores
- UITableView