Swift Optionals

AP CS Principles 2021
Cody Henrichsen

1

What is an Optional

- Designed to prevent inappropriate crashing
- Builds on the Swift design principle that nil is not a desired value 99% of the time!
- All chances for nil must be EXPLICITLY handled by code by design!
- Formally, it is an enumeration that has two possible values
  - None : nil
  - Some : a value of the type

2

Optional Type

- If you want an variable to potentially hold nil it must be explicitly declared as an optional of that Type
- When working with the UIViewController type interfaces, all GUI widgets had to be Optional since the component was not initialized when inflated from the storyboard
- Many methods return Optional Types especially when dealing with files, networks, or permissions
- The Optional is indicated by one of the two operators AFTER the associated method or variable

AKA Postfix notation!

3

Optional Chaining

- ?
  - This is the optional chaining operator. This operator attached to a method or variable indicates that Swift will look at the contents, and if a nil is there the statement will NOT execute. If the value is present then Swift will execute the statement and continue.
  - Most often used in the if let pattern
  - AKA peeking

4

Force Unwrap

- !
  - This is the force unwrap operator. Swift will unwrap the optional and proceed. If it encounters a nil it will crash! If not, Swift will execute the statement and continue

5

Optional Handling: if let

- if let
  - Assigns the value into a let when a value is present and proceed
  - Safely organizes code in executable blocks

```
struct InsideAccess
{
    var name : String
    var potential : Double?
}

let sample = InsideAccess(name: "valid", potential: 4.3)
print (sample.name)
if let interiorValue = sample.potential
{
    print (interiorValue)
}
print ("Done with optional demo")
```

```
InsideAccess
"valid\n"

"4.3\n"

"Done with optional demo
```

6

Optional Handling: Default value

- ??
  - This operator supplies a default value when a nil value is unwrapped. Of course the default value needs to match the Type
  - Official name is the nil-coalescing operator

```
struct InsideAccess
{
    var name : String
    var potential : Double?
}

let sample = InsideAccess(name: "valid")
print (sample.name)
print (sample.potential ?? 0.00 + 3.1415)
```

7

Optional Handling: guard let

- Check that the Optional unwraps and continue on with direct access
- Needs to be used in a method
- Needs an else with a way out of scope AKA return
  - Void means just return
  - Otherwise return the Type specified

```
struct InsideAccess
{
    var name : String
    var potential : Double?
}

func demoWithGuard(parameter : InsideAccess) -> Void
{
    guard let interiorValue = parameter.potential
    else
    {
        print ("nothing inside the potential")
        return
    }

    print (interiorValue)
}

let sample = InsideAccess(name: "valid", potential: 4.3)
print ("sample.name")
demoWithGuard(parameter: sample)
print ("Done with optional demo")
```

```
'4.3'

InsideAccess
'valid'
"Done with optional demo"
```

8

Bonus: Error Handling via Optional

- try?
  - Attempt to execute a method that throws an Error but converts to an Optional handled with Optional Chaining
- try!
  - Attempt the method but if it throws an Error CRASH
  - **DON'T!!** unless you **KNOW** the code/data is safe!

9