SCIENCES SORBONNE UNIVERSITÉ

Swift, optionals

Fabrice.Kordon@lip6.fr

LIP6

Asan introduction…

Similar mechanisms in some (rare) languages

- Notion for «safe no values»
  - **LISP/Scheme: comparison to an «uninitialized variable»**
  - **Caml: «Option» type (None of Somme x | None)**

The need

- What if… computation may return «no value»
  - **Select a value to be «no value»**
  - **Return nothing (impossible in functions)**
  - **Return «nil» (but for classes only)**
  - **Raise an exception**
  - **etc.**

In Swift, a dedicated mechanism

- Objective, use the values if it exists
  - **And avoid crashes otherwise**
- A universal «no value» for all types

A first approach to optionals

Function returning an object

```
func myFunction (val : Int) -> String? {
    if val == 1 {
        return "One"
    } else if val == 2 {
        return "Two"
    }
    return nil // explicitly state "no value"
}

if let val = myFunction(val: 1) {
    print("1 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 2) {
    print("2 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 3) {
    print("3 = \(val)")
} else {
    print("Unknown value")
}
```

A first approach to optionals

Function returning an object

```
func myFunction (val : Int) -> S
    if val == 1 {
        return "One"
    } else if val == 2 {
        return "Two"
    }
    return nil // explicitly state "
}
```

Heeee!

Possible in OO language

```
if let val = myFunction(val: 1) {
    print("1 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 2) {
    print("2 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 3) {
    print("3 = \(val)")
} else {
    print("Unknown value")
}
```

A first approach to optionals

Function returning an object

```
func myFunction (val : Int) -> S
    if val == 1 {
        return "One"
    } else if val == 2 {
        return "Two"
    }
    return nil // explicitly state "
}
```

Heeee!

Possible in OO language

```
if let val = myFunction(val: 1) {
    print("1 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 2) {
    print("2 = \(val)")
} else {
    print("Unknown value")
}
if let val = myFunction(val: 3) {
    print("3 = \(val)")
} else {
    print("Unknown value")
}
```

```
1 = One
2 = Two
Unknown value
```

And now for elementary types


```
func searchInString(str: String, tab: [String]) -> Int? {
    var ind = 0
    for val in tab {
        if val == str {
            return ind
        }
        ind += 1
    }
    return nil
}

let tabstr = ["Grandma", "Titi", "Rominet", "Brutus"]
var s = "Titi"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
}  else {
    print("\(s) absent")
}

s = "Brutor"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
} else {
    print("\(s) absent")
}
```

And now for elementary types


```swift
func searchInString(str: String, tab: [String]) -> Int? {
    var ind = 0
    for val in tab {
        if val == str {
            return ind
        }
        ind += 1
    }
    return nil
}

let tabstr = ["Grandma", "Titi", "Rominet", "Brutus"]
var s = "Titi"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
}  else {
    print("\(s) absent")
}

s = "Brutor"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
} else {
    print("\(s) absent")
}
```

```
Titi found at index 1
Brutor absent
```

And now for elementary types

Works for basic types

The very same mechanism

```
func searchInString(str
    var ind = 0
    for val in tab {
        if val == str {
            return ind
        }
        ind += 1
    }
    return nil
}

let tabstr = ["Grandma", "Titi", "Rominet", "Brutus"]
var s = "Titi"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
}  else {
    print("\(s) absent")
}

s = "Brutor"
if let val = searchInString (str: s, tab: tabstr) {
    print("\(s) found at index \(val)")
} else {
    print("\(s) absent")
}
```

```
Titi found at index 1
Brutor absent
```

Let's provide details

This «nil» means «no value» for any type

- Variables declared as «optional»
- Check of value's existence is required

```
func searchInString(str: String, tab: [String]) -> Int? {
    var ind = 0
    for val in tab {
        if val == str {
            return ind
        }
        ind += 1
    }
    return nil
}

let tabstr = ["Grandma", "Titi", "Rominet", "Brutus"]
let v : Int? = searchInString (str: "Sylvester", tab: tabstr)
if (v != nil) {
    print("We found Sylvester")
} else {
    print("He must be hidden")
}
```

- Useful to report errors
  - **But not a diagnosis**

Let's provide details

This «nil» means «no value» for any type

- Variables declared as «optional»
- Check of value's existence is required

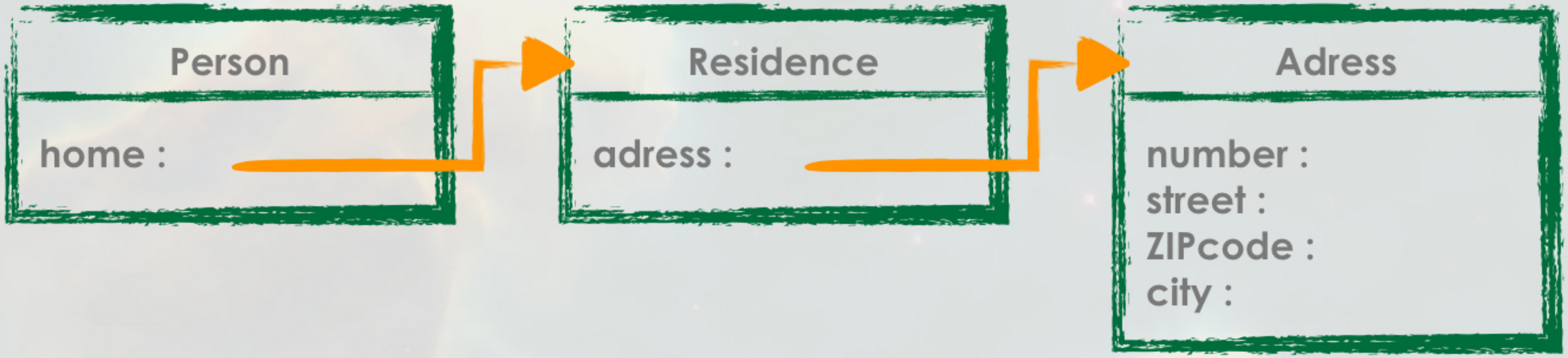
```
func searchInString(str: String, tab: [String]) -> Int? {
    var ind = 0
    for val in tab {
        if val == str {
            return ind
        }
        ind += 1
    }
    return nil
}

let tabstr = ["Grandma", "Titi", "Rominet", "Brutus"]
let v : Int? = searchInString (str: "Sylvester", tab: tabstr)
if (v != nil) {
    print("We found Sylvester")
} else {
    print("He must be hidden")
}
```

- Useful to report errors
  - **But not a diagnosis**

```
He must be hidden
```

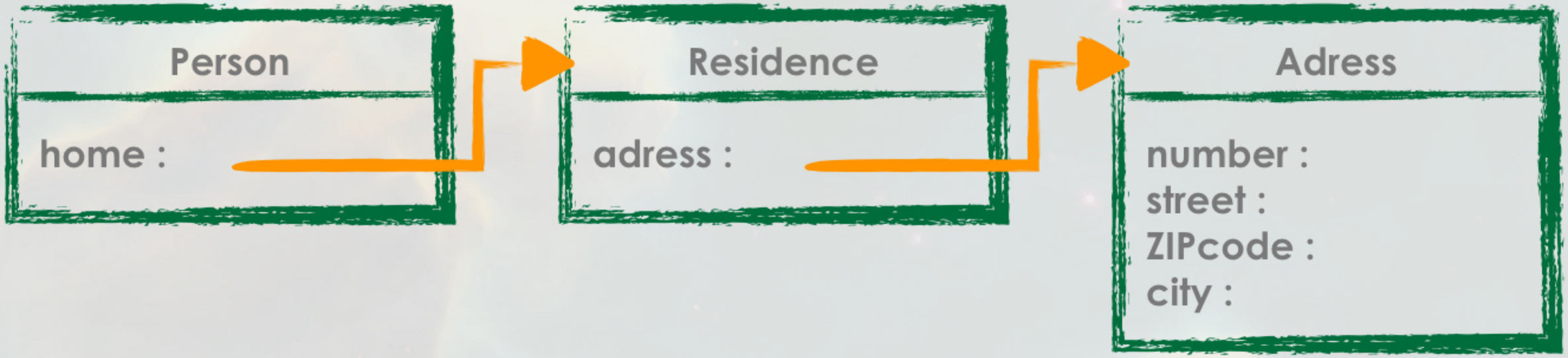
Useful for dereferences



Secure code means «heavy writing»

```
if let h = fabrice.home {
    if let a = h.address {
        if let n = a.number {
            print("number s \(n)")
        }
    }
}
```

Useful for dereferences



Secure code means «heavy writing»

```
if let h = fabrice.home {
    if let a = h.address {
        if let n = a.number {
            print("number s \(n)")
        }
    }
}
```

Possible simplified writing

```
n = fabrice.home?.adress?.number // «nil» if a link misses
if n == nil {
    print("number \(n)")
}
```

As a conclusion…

How could we do without optionals in Objective-C?

- Certainly nothing for basic types…
- But…

… Objective-C le you talk with ghosts!

- These are the premises of this mechanism

One more (important) thing

- When you need for enforce referencing : «!» notation
  - **But a «nil» raises an error**

As a conclusion…

How could we do without optionals in Objective-C?

- Certainly nothing for basic types…
- But…

…ith ghosts!

Make code simpler…

increase its readability!

- Th… …echanism

One more (important) thing

- When you need for enforce referencing : «!» notation
  - **But a «nil» raises an error**