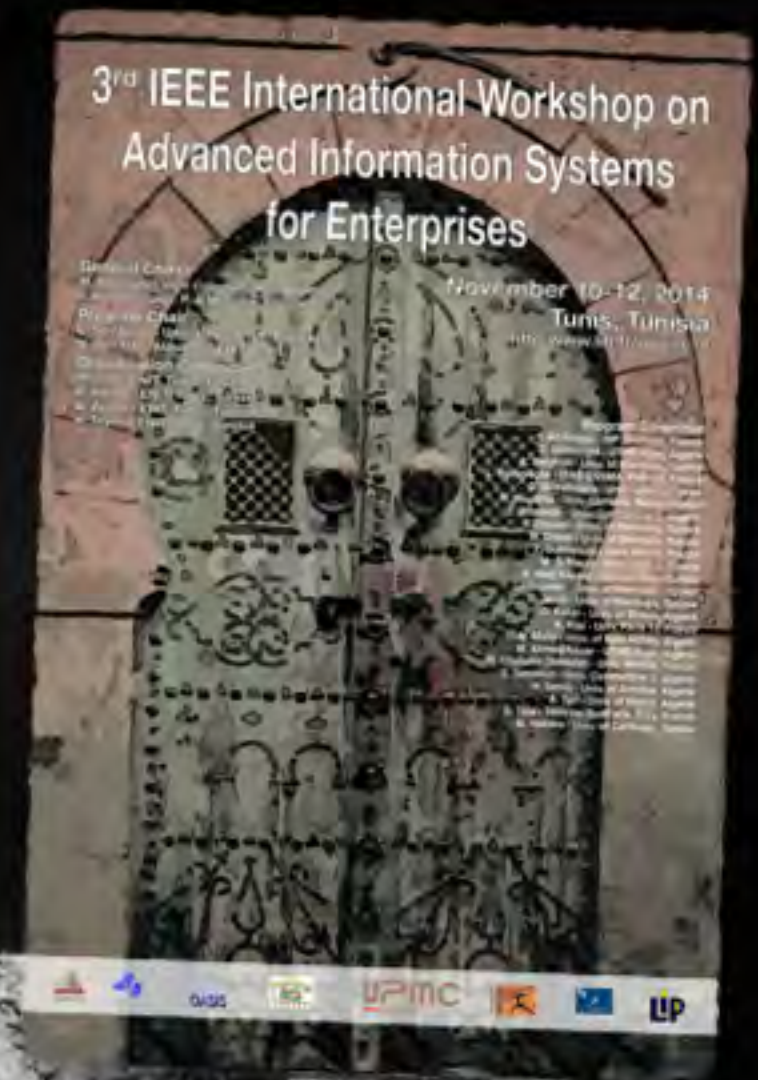


Mobile platforms Programming: iOS (session 2)



Tutorial
IWAISE'2014 — Tunis — November 12, 2014

Prolégomènes (sur Swift)

Fabrice.Kordon@lip6.fr

En guise d'introduction

3

📱 Présentation de Swift délicate...

- Langage stabilisé le 12 septembre (Keynote + Xcode 6 GM)
- Pas mal de modifications entre juin et septembre 2014

📱 Le «truc d'Apple»



«moderne»



«puissant»



«sûr»

Objectif de ces séquences sur Swift

4

- 📱 Vous en donner un aperçu
- 📱 En aucun cas faire de vous des spécialistes
 - La pratique est nécessaire
 - Moi-même je n'ai pas encore assez de recul
- 📱 Vous donnez les moyens de

RTFM!



Swift, les types de base

Fabrice.Kordon@lip6.fr

En guise d'introduction...

6

Vous connaissez déjà

- String, Int, Double, Bool

Regardons rapidement...

- Types énumérés
- Tableaux & Dictionnaires

Immuables, Mutables??

- Système let/var
 - ▶ Attention, ce sera plus subtil pour une classe

Remarque importante (valeurs rendues)

- «type» versus «type?»

Types énumérés (1)

7

C'est une «suite de valeurs»...

```
enum Suite {case trèfle, carreau, cœur, pique}

let typeCarte = Suite.trèfle
let autreTypeCarte = Suite.carreau

if typeCarte == Suite.carreau {
  println ("C'est un carreau");
} else {
  println ("C'est autre chose");
}
println ("et de un : \(typeCarte), et de deux : \(autreTypeCarte)")
```

C'est autre chose
et de un : (Enum Value), et de deux : (Enum Value)

Types énumérés (2)

8

C'est une «suite de valeurs»...

- Potentiellement associée à des fonctions

```
enum Suite {case trèfle, carreau, cœur, pique
  func image() -> String {
    switch self {
    case .trèfle:
      return "Trèfle"
    case .cœur:
      return "Cœur"
    case .pique:
      return "Pique"
    case .carreau:
      return "Carreau"
    }
  }
}
```

```
let v1 = Suite.trèfle
let v2 = Suite.carreau
println ("et de un : \(v1.image()), et de deux : \(v2.image())")
```

et de un : Trèfle, et de deux : Carreau

Types énumérés (3)

C'est une suite de valeurs...

- Associable à des entiers

```
enum Suite : Int {case trèfle = 1, carreau, cœur, pique}
```

```
let v1 = Suite.trèfle
```

```
var pos = 3
```

```
let v2 = Suite.fromRaw(pos)
```

```
println ("et de un : \(v1.toRaw())")
```

```
if v2 == Suite.cœur {
```

```
    println ("Cœur est bien la 3eme couleur!")
```

```
}
```

```
pos = 5
```

```
if Suite.fromRaw(pos) == Suite.pique {
```

```
    println("Pique est bien est bien la 5eme couleur")
```

```
} else {
```

```
    println("La cata!!!")
```

```
}
```

et de un : 1

Cœur est bien la 3eme couleur!

La cata!!!

Types énumérés (3)

9

C'est une suite de valeurs...

- Associable à des entiers

```
enum Suite : Int {case trèfle = 1, carreau, cœur, pique}
```

```
let v1 = Suite.trèfle
```

```
var pos = 3
```

```
let v2 = Suite.fromRaw(pos)
```

```
println ("et de un : \"(v1.toRaw())\"")
```

```
if v2 == Suite.cœur {
```

```
    println ("Cœur est bien la 3eme couleur!")
```

```
}
```

```
pos = 5
```

```
if Suite.fromRaw(pos) == Suite.pique {
```

```
    println("Pique est bien est bien la 5eme couleur")
```

```
} else {
```

```
    println("La cata!!!")
```

```
}
```

Peut rendre
«nil»

et de un : 1

Cœur est bien la 3eme couleur!

La cata!!!

Types énumérés (3)

9

📱 C'est une suite de valeurs...

- Associable à des entiers

```
enum Suite : Int {case trèfle = 1, carreau, cœur, pique}
```

```
let v1 = Suite.trèfle
```

```
var pos = 3
```

```
let v2 = Suite.fromRaw(pos)
```

```
println ("et de un : \(v1.toRaw())")
```

```
if v2 == Suite.cœur {
```

```
    println ("Cœur est bien la 3eme couleur!")
```

```
}
```

```
pos = 5
```

```
if Suite.fromRaw(pos)
```

```
    println("Pique est la 5eme couleur")
```

```
} else {
```

```
    println("La cata!!!")
```

```
}
```

Peut rendre
«nil»

RTFM!



et de un : 1
Cœur est bien la 3eme couleur!
La cata!!!

Tableaux

10

C'est «fastoche»...

```
var tab = ["Titi", "Rominet", "Grand-mère"]
tab.append("Brutus") // ne marcherait pas avec "let"
println("tab = \(tab)")
tab.removeAtIndex(1)
if tab.isEmpty {
    println("tableau vide")
} else {
    println("tab = \(tab)")
}
println ("Le chien s'appelle \(tab[2])")

var t2 = [Int]()
t2.append(0)
t2[0] = 1 // ne fonctionnerait pas avec "let"
t2 += [3, 5, 7, 11] // ne fonctionnerait pas non plus avec "let"
println("Voyons ces \(t2.count) nombres premiers: \(t2)")
```

tab = [Titi, Rominet, Grand-mère, Brutus]

tab = [Titi, Grand-mère, Brutus]

Le chien s'appelle Brutus

Voyons ces 5 nombres premiers: [1, 3, 5, 7, 11]

Tableaux

10

📱 C'est «fastoche»... 😏

```
var tab = ["Tit", "Romet", "Grand-mère"]
tab.append("Brutus") // ne marcherait pas avec "let"
println("tab = \(tab)")
tab.removeAtIndex(1)
if tab.isEmpty {
    println("tableau vide")
} else {
    println("tab = \(tab)")
}
println ("Le chien s'appelle \(tab[2])")
```

```
var t2 = [Int]()
t2.append(0)
t2[0] = 1 // ne fonctionne pas avec "let"
t2 += [3, 5, 7, 11] // ne fonctionne pas non plus avec "let"
println("Voyons ces 5 nombres premiers: \(t2)")
```

RTFM!



tab = [Tit, Romet, Grand-mère, Brutus]

tab = [Tit, Grand-mère, Brutus]

Le chien s'appelle Brutus

Voyons ces 5 nombres premiers: [1, 3, 5, 7, 11]

Dictionnaires

11

📱 Presque aussi facile que les tableaux... 😏

```
var cc = [ "Trèfle" : "Noire",  
          "Carreau" : "Rouge",  
          "Cœur" : "Rouge",  
          "Pique" : "Rouge"]  
println("Dictionnaire de \((cc.count)\) éléments = \((cc)\)")  
cc["Pique"] = "Noire"  
println("Dictionnaire = \((cc)\)")  
for(key,value) in cc  
{  
    println("key: \((key)\) value:\((value)\)")  
}  
for key in cc.keys {  
    println("Key: \((key)\)")  
}  
for values in cc.values {  
    println("Value: \((values)\)")  
}  
let s1 = cc["Trèfle"]  
let s2 = cc["Toto"]  
println("s1 = \((s1)\) et s2 =\((s2)\)")
```

Dictionnaire de 4 éléments = [Trèfle: Noire,
Cœur: Rouge, Pique: Rouge, Carreau:
Rouge]

Dictionnaire = [Trèfle: Noire, Cœur: Rouge,
Pique: Noire, Carreau: Rouge]

key: Trèfle value:Noire

key: Cœur value:Rouge

key: Pique value:Noire

key: Carreau value:Rouge

Key: Trèfle

Key: Cœur

Key: Pique

Key: Carreau

Value: Noire

Value: Rouge

Value: Noire

Value: Rouge

s1 = Optional("Noire") et s2 =nil



Dictionnaires

11

📱 Presque aussi facile que les tableaux... 😊

```
var cc = [ "Trèfle" : "Noire",  
          "Carreau" : "Rouge",  
          "Cœur" : "Rouge",  
          "Pique" : "Rouge"]  
println("Dictionnaire de \((cc.count)\) éléments = \((cc)\)")  
cc["Pique"] = "Noire"  
println("Dictionnaire = \((cc)\)")  
for(key,value) in cc  
{  
    println("key: \((key)\) value:\((value)\)")  
}  
for key in cc.keys {  
    println("Key: \((key)\)")  
}  
for values in cc.values {  
    println("Value: \((values)\)")  
}  
let s1 = cc["Trèfle"]  
let s2 = cc["Toto"]  
println("s1 = \((s1)\) et s2 =\((s2)\) ,
```

Dictionnaire de 4 éléments = [Trèfle: Noire,
Cœur: Rouge, Pique: Rouge, Carreau:
Rouge]

Dictionnaire = [Trèfle: Noire, Cœur: Rouge,
Pique: Noire, Carreau: Rouge]

key: Trèfle value:Noire
key: Cœur value:Rouge
key: Pique value:Noire
key: Carreau value:Rouge

Key: Trèfle
Key: Cœur
Key: Pique
Key: Carreau

Value: Noire
Value: Rouge
Value: Noire
Value: Rouge

s1 = Optional("Noire") et s2 =nil



En guise de conclusion...

12

📱 Vous avez déjà de quoi faire!!!

📱 Et bien des choses à découvrir dans le **RTFM!**



Swift, structures de contrôle

Fabrice.Kordon@lip6.fr

En guise d'introduction

14

Structures de contrôle?

- Objectif, contrôler le traitement de données

Structures classiques

- Conditionnelles
- Boucles
- «case»

► mais là il y a «boost»



Conditionnelles

15

Du très classique...

```
let str = "Rominet"
if str == "Rominet" {
    println ("Titi est en danger!!!")
} else {
    println ("Tout est calme")
}

var note = 13
if note < 10 {
    println("Examen raté")
} else if (note >= 10) && (note < 14) {
    println("C'est gagné")
} else {
    println("Dans le peloton de tête")
}
```

Titi est en danger!!!
C'est gagné

Boucles énumératives

16

Approche très «à la Ada»

- Indice déclaré de manière implicite
- Accès dans le corps de la boucle en lecture seulement

Utile pour parcourir les tableaux

- Et les dictionnaires aussi...

```
for i in 1 ... 5 {  
    print("i = \(i), ")  
}  
println("fin")
```

```
let telluriques = [ "Mercure", "Venus", "Terre", "Mars"]  
for p in telluriques {  
    print ("\(p), ")  
}
```

i = 1, i = 2, i = 3, i = 4, i = 5, fin
Mercure, Venus, Terre, Mars,

Boucles avec condition et incrément explicite

17

📱 Super classique!!!

```
for var i = 0; i < 3; ++i {  
    println("i = \$(i)")  
}
```

i = 0

i = 1

i = 2

Boucles «while» (test en début ou fin de boucle)

18

```
var x = 3
while x < 5 {
    println ("x = \(x)")
    x++
}
```

```
var y = 3
do {
    println ("y = \(y)")
    y++
} while y < 5
```

x = 3

x = 4

y = 3

y = 4

Switch...

```
enum PlatnetesTelluriques {case Mercure, Venus, Terre, Mars }
var p = PlatnetesTelluriques.Terre

switch p {
case .Terre :
    println("C'est habitable")
case .Mars :
    println ("Il y fait trop froid");
default:
    println ("Il y fait beaucoup trop chaud")
}
```

Switch...

```
enum PlanetesTelluriques {case Mercure, Venus, Terre, Mars }  
var p = PlanetesTelluriques.Terre  
  
switch p {  
case .Terre :  
    println("C'est habitable")  
case .Mars :  
    println ("Il y fait trop froid");  
default:  
    println ("Il y fait beaucoup trop chaud")  
}
```

Toutes les valeurs
doivent être couvertes!!!

En guise de conclusion...

20

📱 Rien de bien compliqué...

📱 Mais des structures saines

- Fermetures obligatoires (contrairement au C)
- Couverture des domaines de valeurs
- Protection des variables d'indice...



📱 Retours d'expérience sur la «programmation saine»

- Typage,
- Constructions syntaxiques du langage
- etc.

Swift, tuples et filtrage

Fabrice.Kordon@lip6.fr

En guise d'introduction...

22

Mécanismes originaux

Tuple = Regroupement temporaires de valeurs

- Similaire à la notion d'agrégat en Ada

▶ Dictionnaires...

```
for(key,value) in cc {  
    println("key: \ (key) value:\ (value)")  
}
```

- Possibilité de faire du «filtrage» («pattern matching»)

▶ Puissance issue de mécanismes propres aux langages fonctionnels

Voyons donc

Les tuples par l'exemple...

23



Construction

```
let http404pb = (404, "page non trouvée")
println ("regardons ce que cela donne : \(http404pb)")
```



Séparation

```
let (code, msg) = http404pb
println ("code      = \(code)")
println ("message   = \(msg)")
```



Extraction (avec et sans nommage)

```
println ("juste le message = \(http404pb.1)")
println ("juste le code    = \(http404pb.0)")
```

```
let http404pb2 = (code : 404, desc : "page non trouvée")
println ("juste le coe      = \(http404pb2.code)")
println ("juste le message = \(http404pb2.desc)")
```

Tuples et filtrage (premier aperçu)

24

Le mécanisme est puissant!!!

```
func localise (point : (Int, Int)) -> String {  
  switch point {  
    case (0, 0):  
      return "(0, 0) est à l'origine"  
    case (_, 0):  
      return "(\"(point.0), 0) est sur l'axe des X")  
    case (0, _):  
      return "(0, \"(point.1)) est sur l'axe des Y")  
    case (-2...2, -2...2):  
      return "(\"(point.0), \"(point.1)) est dans la boite")  
    default:  
      return "(\"(point.0), \"(point.1)) est hors de la boite")  
  }  
}  
...
```

Tuples et filtrage (premier aperçu)

24

Le mécanisme est puissant!!!

```
let pt0 = (0, 0)
let pt1 = (1, 1)
let pt2 = (1, 0)
let pt3 = (0, 1)
let pt4 = (3, 1)
let pt5 = (3, 0)
```

```
println "\(pt0) => \(localise(pt0))"
println "\(pt1) => \(localise(pt1))"
println "\(pt2) => \(localise(pt2))"
println "\(pt3) => \(localise(pt3))"
println "\(pt4) => \(localise(pt4))"
println "\(pt5) => \(localise(pt5))"
```

```
func localise (point : (Int, Int)) -> String {
  switch point {
    case (0, 0):
      return "(0, 0) est à l'origine"
    case (_, 0):
      return "(\(point.0), 0) est sur l'axe des X"
    case (0, _):
      return "(0, \(point.1)) est sur l'axe des Y"
    case (-2...2, -2...2):
      return "(\(point.0), \(point.1)) est dans la boîte"
    default:
      return "(\(point.0), \(point.1)) est hors de la boîte"
  }
}
```

(0, 0) => (0, 0) est à l'origine
(1, 1) => (1, 1) est dans la boîte
(1, 0) => (1, 0) est sur l'axe des X
(0, 1) => (0, 1) est sur l'axe des Y
(3, 1) => (3, 1) est hors de la boîte
(3, 0) => (3, 0) est sur l'axe des X

Filtrage et conditions sophistiquées

25

Avec nommage local des composants du tuple...

```
func diagonales (point : (Int, Int)) -> String {  
  switch point {  
  case let (x, y) where x == y:  
    return ("(\(x), \(y)) est sur la diagonale")  
  case let (x, y) where x == -y:  
    return ("(\(x), \(y)) est sur l'autre diagonale")  
  case let (x, y):  
    return ("(\(x), \(y)) n'est sur aucune diagonale")  
  }  
}
```

```
let pt6 = (3, -3); let pt7 = (2, 2); let pt8 = (3, 1);
```

```
println ("\(pt6) => \(diagonales(pt6))")  
println ("\(pt7) => \(diagonales(pt7))")  
println ("\(pt8) => \(diagonales(pt8))")
```

(3, -3) => (3, -3) est sur l'autre diagonale
(2, 2) => (2, 2) est sur la diagonale
(3, 1) => (3, 1) n'est sur aucune diagonale

En guise de conclusion...

26

📱 Ceci n'est qu'un aperçu des possibilités

- Elles sont nombreuses

📱 Plus où vous savez

RTFM!



Swift, les «optionals»

Fabrice.Kordon@lip6.fr

Introduction



Le besoin

- Que faire lorsqu'un calcul peut ne rendre aucune valeur?
 - ▶ Décider d'une valeur «pas de valeur»
 - ▶ Ne rien rendre (impensable en retour d'une fonction)
 - ▶ Renvoyer «nil» (pour les classes)
 - ▶ Lever une erreur ou une exception
 - ▶ etc.



Il faut disposer d'un mécanisme dédié

- Profiter de la valeur si elle existe
- Ne pas planter dans le cas contraire
- Disposer d'un «pas de valeur» universel...

Première approche des «optionals»...

29



Fonction retournant une classe...

```
func maFonction (val : Int) -> String? {  
    if val == 1 {  
        return "Un"  
    } else if val == 2 {  
        return "Deux"  
    }  
    return nil  
}
```

Première approche des «optionals»...

29

Fonction retournant une classe...

```
if let val = maFonction(1) {  
    println("1 = \ \(val)")  
} else {  
    println("Valeur non traitée")  
}  
  
if let val = maFonction(2) {  
    println("2 = \ \(val)")  
} else {  
    println("Valeur non traitée")  
}  
  
if let val = maFonction(3) {  
    println("3 = \ \(val)")  
} else {  
    println("Valeur non traitée")  
}
```

```
func maFonction (val : Int) -> String? {  
    if val == 1 {  
        return "Un"  
    } else if val == 2 {  
        return "Deux"  
    }  
    return nil  
}
```

1 = Un
2 = Deux
Valeur non traitée

Et pour les types élémentaires?

30

Cela fonctionne aussi

```
func chercheDansChaine(str: String, array: [String]) -> Int? {  
    for (ind, val) in enumerate(array) {  
        if val == str {  
            return ind  
        }  
    }  
    return nil  
}
```

Et pour les types élémentaires?

30

Cela fonctionne aussi

```
func chercheDansChaine(str: String, array: [String]) -> Int? {  
    for (ind, val) in enumerate(array) {  
        if val == str {  
            return ind  
        }  
    }  
    return nil  
}  
  
let tabstr = ["Grand-mère", "Titi", "Rominet", "Brutus"]  
var s = "Titi"  
  
if let val = chercheDansChaine (s, tabstr) {  
    println("\(s) trouvé en position \(val)")  
} else {  
    println("\(s) absent")  
}  
  
s = "Brutor"  
if let val = chercheDansChaine (s, tabstr) {  
    println("\(s) trouvé en position \(val)")  
} else {  
    println("\(s) absent")  
}
```

Titi trouvé en position 1
Brutor absent

Mais encore...



Ce «nil» signifie «pas de valeur» pour tous les types...

- Il faut déclarer le type en «optionals»
- Et ensuite tester si la valeur existe

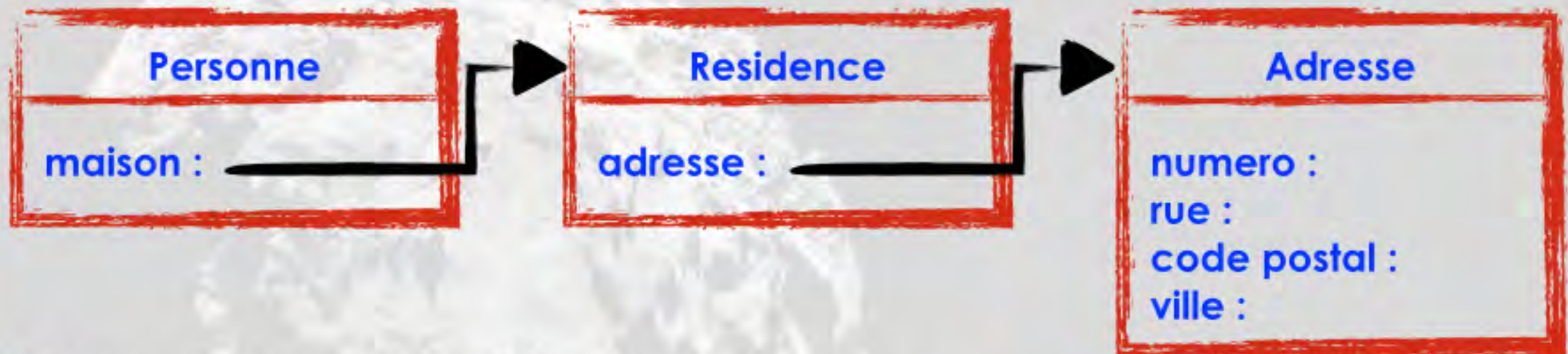
```
let v : Int? = chercheDansChaine ("Sylvestre", tabstr)

if (v != nil) {
    println("On a trouvé Sylvestre")
} else {
    println («Il a du se déguiser»)
}
```

- Cela peut servir à remonter des erreurs
 - ▶ Mais pas un diagnostic...

Utilisation dans les références entre classes

32

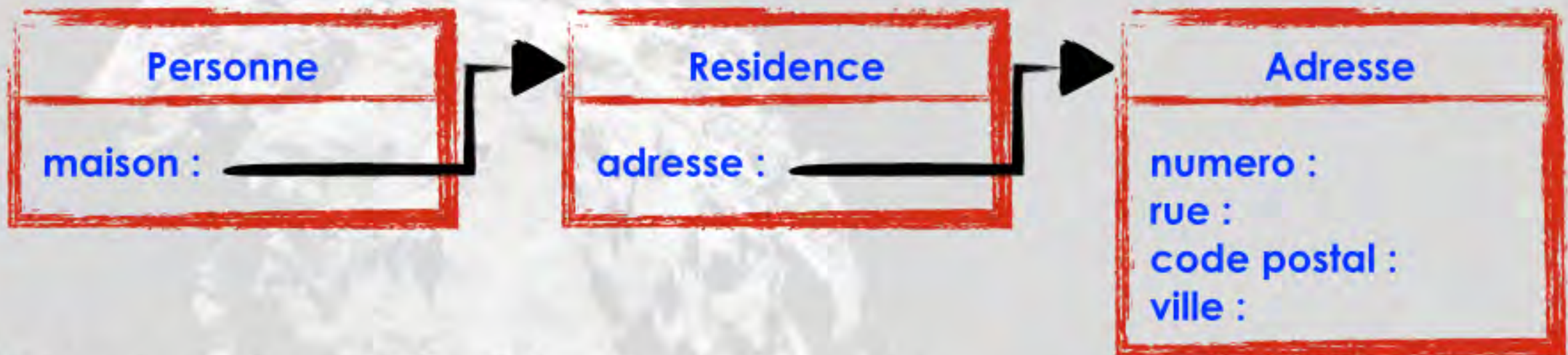


Code sécurisé = écriture lourde

```
if let h = fabrice.maison {  
  if let a = h.adresse {  
    if let n = a.numero {  
      println ("numéro \n")  
    }  
  }  
}
```

Utilisation dans les références entre classes

32



Code sécurisé = écriture lourde

```
if let h = fabrice.maison {  
  if let a = h.adresse {  
    if let n = a.numero {  
      println ("numéro \n(n)")  
    }  
  }  
}
```

Écriture simplifiée...

```
n = fabrice.maison?.adresse?.numero // «rien» si problème de liaison
```

En guise de conclusion...

33

Mais comment faisait-on en Objective-C?

- Rien pour les types simples...
- Mais...

... on pouvait causer à des fantômes



- Donc on disposait des prémisses de ce mécanisme finalement

Par contre...

- Notation «!» pour forcer un chaînage via un pointeur
 - ▶ À n'utiliser que si on est certain de ne pas trouver «nil» quelque part!!!

En guise de conclusion...

33

📱 Mais comment faisait-on en Objective-C?

- Rien pour les types simples...
- Mais...

... on pouvait causer à des fantômes



- Donc on dit

📱 Par contre

- Notation «!» pour forcer un en-tête

▶ À n'utiliser que si on est certain de ne pas trouver ailleurs quelque part!!!

**Un tel mécanisme accroît
la fiabilité des programmes**



UPMC

SORBONNE UNIVERSITÉS

Swift, écrire ses propres classes

Fabrice.Kordon@lip6.fr

En guise d'introduction...

📱 Structure d'une classe:

- Interface + implémentation **dans un seul fichier**

📱 Syntaxe «rénovée»

- Inspirée des langages objets (of course)
- Inspirée des langages fonctionnels
- Contient
 - ▶ Des «propriétés» (comme Objective-C)
 - ▶ Des «méthodes»
 - ▶ Des «initialiseurs»
 - ▶ Gestion de la «finalisation» (deinit)

Exemple 1 — Student

36

```
class Student {  
    var numero = 0  
    var nom = ""  
  
    func identite () -> String {  
        return "\("nom) (\(numero))"  
    }  
}
```



Getter/Setters déclarés automatiquement

- Système var/let détermine la présence du setter
- On peut aussi les surcharger (attributs calculés)

Exemple 1 — Student

36

```
class Student : C {  
    var numero = 0  
    var nom = ""  
  
    func identite () -> String {  
        return "\(nom) (\(numero))"  
    }  
}
```

Si héritage...

📱 Getter/Setters déclarés automatiquement

- Système var/let détermine la présence du setter
- On peut aussi les surcharger (attributs calculés)

Exemple 1 — Student

36

```
class Student :C {
```

```
    var numero = 0
```

```
    var nom = ""
```

```
    func identite () -> String {  
        return "\(nom) (\(numero))"  
    }
```

```
}
```

```
let s1 = Student()
```

```
s1.nom = "Tartempion"
```

```
s1.numero = 1000
```

```
println ("Voici \(s1.nom) avec le numéro \(s1.numero)")
```

Si héritage...

Voici Tartempion avec le numéro 1000

📱 Getter/Setters déclarés automatiquement

- Système var/let détermine la présence du setter
- On peut aussi les surcharger (attributs calculés)

Exemple 1 — Student + attributs privés

37

```
class Student {  
  private var nom = ""  
  private var numero = 0  
  
  init (nm : String, nu : Int) {  
    nom = nm  
    numero = nu  
  }  
  
  func identite () -> String {  
    return "\(nom) (\(numero))"  
  }  
  
  func Nom() -> String {  
    return nom  
  }  
  
  func Numero() -> Int {  
    return numero  
  }  
}
```

Exemple 1 — Student + attributs privés

37

```
class Student {  
  private var nom = ""  
  private var numero = 0  
  
  init (nm : String, nu : Int) {  
    nom = nm  
    numero = nu  
  }  
  
  func identite () -> String {  
    return "\(nom) (\(numero))"  
  }  
  
  func Nom() -> String {  
    return nom  
  }  
  
  func Numero() -> Int {  
    return numero  
  }  
}
```

Voici Tartempion avec le numéro 1000

```
let s1 = Student(nm : "Tartempion", nu : 1000)  
println ("Voici \(s1.Nom()) avec le numéro \(s1.Numero())")
```

Exemple 1 — Student + attributs privés

37

```
class Student {  
  private var nom = ""  
  private var numero = 0  
  
  init (nm : String, nu : Int) {  
    nom = nm  
    numero = nu  
  }  
  
  func identite () -> String {  
    return "\(nom) (\(numero))"  
  }  
  
  func Nom() -> String {  
    return nom  
  }  
  
  func Numero() -> Int {  
    return numero  
  }  
}
```

On pourrait aussi
faire les setter

Voici Tartempion avec le numéro 1000

```
let s1 = Student(nm : "Tartempion", nu : 1000)  
println ("Voici \(s1.Nom()) avec le numéro \(s1.Numero())")
```

Exemple 1 — Student + attributs privés

37

```
class Student {  
  private var nom = ""  
  private var numero = 0  
  
  init (nm : String, nu : Int) {  
    nom = nm  
    numero = nu  
  }  
  
  func identite () -> String {  
    return "\(nom) (\(numero))"  
  }  
  
  func Nom() -> String {  
    return nom  
  }  
  
  func Numero() -> Int {  
    return numero  
  }  
}
```

On pourrait aussi
faire les setter

private : dans le fichier
public : n'importe où
internal : dans un module

Voici Tartempion avec le numéro 1000

```
let s1 = Student(nm : "Tartempion", nu : 1000)  
println ("Voici \(s1.Nom()) avec le numéro \(s1.Numero())")
```

Quelques remarques sur les attributs

38

S'ils sont déclarés avec une valeur par défaut

- Pas besoin de fonction init pour les initialiser...
 - ▶ Mais rien n'empêche de l'y mettre

S'ils sont juste déclarés

- Il **faut** une fonction d'initialisation

Le compilateur le rappelle



Attributs calculés

```
class Student {  
    private var nom = ""  
    private var numero = 0  
  
    init (nm : String, nu : Int) {  
        nom = nm  
        numero = nu  
    }  
  
    var identite : String {  
        get {  
            return "\($nom) (\($numero))"  
        }  
    }  
  
    func Nom() -> String {  
        return nom  
    }  
  
    func Numero() -> Int {  
        return numero  
    }  
}
```

Attributs calculés

```
class Student {  
    private var nom = ""  
    private var numero = 0  
  
    init (nm : String, nu : Int) {  
        nom = nm  
        numero = nu  
    }  
  
    var identite : String {  
        get {  
            return "\($nom) (\($numero))"  
        }  
    }  
  
    func Nom() -> String {  
        return nom  
    }  
  
    func Numero() -> Int {  
        return numero  
    }  
}
```

Voici Tartempion (1000)

```
let s1 = Student(nm : "Tartempion", nu : 1000)  
println ("Voici \($s1.identite)")
```

Attributs «lazy»

40



Attribut dont l'évaluation est paresseuse

- Calcul de la valeur au moment de l'accès seulement
- Ces attributs sont en «var» uniquement
- Ne peut être utilisé sur les attributs calculées

```
class DataImporter {  
    // Importe des données depuis un fichier externe, long à initialiser  
    var fileName = "data.txt"  
    // fonctions d'importation des données  
}  
  
class DataManager {  
    lazy var importer = DataImporter()  
    var data = [String]()  
    // fonctions de gestion des données  
}  
  
let manager = DataManager()  
manager.data.append("Some data")  
manager.data.append("Some more data")  
// l'importateur n'a pas encore été créé même si le «manager» existe déjà
```

Comparaison de références...

41

📱 Des «=» partout



📱 =

- Affectation

📱 === & !==

- Comparaison des deux références
- «est identique à»

📱 Pour rappel, == & !=

- Comparaison de valeurs (e.g. zone mémoires)
- «égal à»

Comparaisons de références — un exemple

42

```
let s1 = Student(nm : "Tartempion", nu : 1000)
let s2 = Student(nm : "Tartempion", nu : 1000)
let s3 = s1
if s1 === s2 {
    println ("C'est la même référence")
} else {
    println ("Références différentes")
}
if s1 === s3 {
    println ("C'est la même référence")
} else {
    println ("Références différentes")
}
```

Références différentes
C'est la même référence

En guise de conclusion...



Les possibilités sur les classes en Swift

- Définir des propriétés (attributs) et des méthodes
- Gérer la visibilité des propriétés
- Surcharger les getter/setter
- Définir des attributs calculés
- Gérer l'évaluation des attributs
- Comparer des références de manière explicite dans le code



Voila pour un début **RTFM!**

- Pour le reste...



Swift, à propos de l'héritage

Fabrice.Kordon@lip6.fr

En guise d'introduction...

45

📱 Swift est un langage objet classique

- Donc il supporte l'héritage...

📱 Héritage «simple»

- Mais couplage avec la généricité («template» en C++)
- Pas d'héritage multiple...

Un exemple simple...

```
class StudentPlus : Student {  
    private var binome : StudentPlus?  
  
    func setBinome (b : StudentPlus) {  
        binome = b  
    }  
  
    func getBinome () -> StudentPlus {  
        return binome! // je suis certain de vouloir la référence  
    }  
}
```

Un exemple simple...

46

```
class StudentPlus : Student {  
    private var binome : StudentPlus?  
  
    func setBinome (b : StudentPlus) {  
        binome = b  
    }  
  
    func getBinome () -> StudentPlus {  
        return binome! // je suis certain de vouloir la référence  
    }  
}  
  
let s1 = StudentPlus(nm : "Toto", nu : 1000)  
let s2 = StudentPlus(nm : "Tata", nu : 1001)  
  
s1.setBinome(s2)  
s2.setBinome(s1)  
  
println ("\"(s1.Nom()) est binômé avec \"(s1.getBinome().Nom())")  
println ("\"(s2.Nom()) est binômé avec \"(s2.getBinome().Nom())")
```

Toto est binômé avec TaTa
Tata est binômé avec Toto

Et l'initialisation?

📱 Attention, appel à **super.init** après

- Contrairement à Objective-C

```
enum Couleur {case rouge, noir, jaune}

class Voiture {
  var peinture: Couleur
  init(coul: Couleur) {
    peinture = coul
  }
}

class voitureDeCourse: Voiture {
  var aUnTurbo: Bool

  init(coul: Couleur, turbo: Bool) {
    aUnTurbo = turbo
    super.init(coul: coul) // l'inverse d'objective-C
  }
}
```

Délégation de l'initialisation...

48

«Convenience» versus «désignée»

- Références stricte (hiérarchisation des appels de super.init)

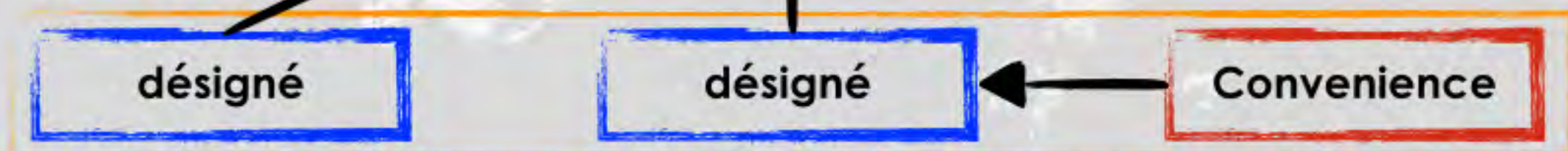
Super super classe



Super classe



Classe



Un exemple d'initialisation «de conveniencence»

49

```
class voitureDeCourse: Voiture {  
    var aUnTurbo: Bool  
  
    init(coul: Couleur, turbo: Bool) {  
        aUnTurbo = turbo  
        super.init(coul: coul)  
        // l'inverse d'objective-C  
    }  
  
    convenience override init (coul: Couleur) {  
        self.init(coul: coul, turbo : true)  
    }  
  
    convenience init () {  
        self.init(coul: .noir)  
    }  
}
```



convenience

- Méthode «pratique»

Un exemple d'initialisation «de conveniencence»

49

```
class voitureDeCourse: Voiture {  
    var aUnTurbo: Bool  
  
    init(coul: Couleur, turbo: Bool) {  
        aUnTurbo = turbo  
        super.init(coul: coul)  
        // l'inverse d'objective-C  
    }  
  
    convenience override init (coul: Couleur) {  
        self.init(coul: coul, turbo : true)  
    }  
  
    convenience init () {  
        self.init(coul: .noir)  
    }  
}
```

convenience

- Méthode «pratique»

Override

- Si recouvrement d'une méthode existante

Un exemple d'initialisation «de conveniencence»

49

```
class voitureDeCourse: Voiture {  
    var aUnTurbo: Bool  
  
    init(coul: Couleur, turbo: Bool) {  
        aUnTurbo = turbo  
        super.init(coul: coul)  
        // l'inverse d'objective-C  
    }  
  
    convenience override init (coul: Couleur) {  
        self.init(coul: coul, turbo : true)  
    }  
  
    convenience init () {  
        self.init(coul: .noir)  
    }  
}
```

convenience

- Méthode «pratique»

Override

- Si recouvrement d'une méthode existante

Appels hiérarchisés

- Seul le «init désigné» fait appel à super.init

Un exemple d'initialisation «de conveniencence»

49

```
class voitureDeCourse: Voiture {  
    var aUnTurbo: Bool  
  
    init(coul: Couleur, turbo: Bool) {  
        aUnTurbo = turbo  
        super.init(coul: coul)  
        // l'inverse d'objective-C  
    }  
  
    convenience override init (coul: Couleur) {  
        self.init(coul: coul, turbo : true)  
    }  
  
    convenience init () {  
        self.init(coul: .noir)  
    }  
}
```

convenience

- Méthode «pratique»

Override

- Si recouvrement d'une méthode existante

Appels hiérarchisés

- Seul le «init désigné» fait appel à super.init

Un exemple d'initialisation «de conveniencence»

49

```
class voitureDeCourse: Voiture {  
    var aUnTurbo: Bool  
  
    ▶ init(coul: Couleur, turbo: Bool) {  
        aUnTurbo = turbo  
        super.init(coul: coul)  
        // l'inverse d'objective-C  
    }  
  
    ▶ conveniencence override init (coul: Couleur) {  
        self.init(coul: coul, turbo : true)  
    }  
  
    ▶ conveniencence init () {  
        self.init(coul: .noir)  
    }  
}
```

📱 conveniencence

- Méthode «pratique»

📱 Override

- Si recouvrement d'une méthode existante

📱 Appels hiérarchisés

- Seul le «init désigné» fait appel à super.init

En guise de conclusion...

50

📱 Des mécanismes assez classiques

📱 Tout ce qu'il faut bien gérer la mémoire

- Construire des applications mieux sécurisées
- Rappelez-vous
 - ▶ L'enjeu c'est l'écosystème
 - ▶ L'écosystème est enrichi par des programmeurs
 - ▶ Donc il faut faciliter le travail de programmation
 - ▶ C'est le compilateur qui travaille
- By the way...
 - ▶ L'enjeu est général
 - ▶ Même raison pour la création du HOLWG en 1975 (origine de Ada)

Swift, gestion de la mémoire & cohérence globale

Fabrice.Kordon@lip6.fr

En guise d'introduction...

52

Avec Swift, la gestion mémoire est (presque) gratuite!

Comment ça marche?

- Blocs (approche FIFO)
 - ▶ Procédures/fonctions, etc
- Classes (exploitation des informations)
 - ▶ Attributs mutables et immuables
 - ▶ Classification des attributs
 - ▶ Constructeur / Destructeur
- Fermetures («closure»)
 - ▶ Gestion du «scope»

ARC est toujours présent!!!

- Il est simplement toujours activé
- Pas de gestion manuelle



Lors de l'initialisation d'un objet

53

Phase 1

- Appel d'un «initialiseur» (désigné ou de convenance)
- Allocation mémoire pour la nouvelle instance
- Confirmation (via l'«initialiseur») que les propriétés sont bien initialisées
- Appel de l'«initialiseur» de la superclasse (jusqu'au «sommet»)
 - ▶ La mémoire de l'instance est alors considérée comme valide

Phase 2

- Finalisation de l'initialisation (self est maintenant accessible)
- Finalisation de via les méthodes «de convenance» s'il y a lieu

Des vérifications sont faites comme

- ▶ Initialisation «locale» avant appel de la superclasse,
- ▶ Appel à la superclasse avant d'affecter des valeurs aux propriétés héritées
- ▶ Pas d'appel à self avant la fin de la phase 1

Retour sur la gestion des liaisons entre objets

54

Différents types de relation

- Strong (relation forte)
 - ▶ Une référence de ce type maintient l'object pointé «vivant»
 - ▶ Attention aux cycles qui risquent de débrayer ARC
 - ▶ Mode par défaut
- Weak (relation faible)
 - ▶ Une référence de ce type ne maintient pas l'object vivant
 - ▶ Type «conditionnel»
 - ▶ Permet de «casser» un cycle
 - ▶ Mot clef : weak
- Unowned
 - ▶ Weak...
 - ▶ Dispose toujours d'une valeur
 - ▶ Mot clef : unowned

Retour sur la gestion des liaisons entre objets

54

Différents types de relation

- Strong (relation forte)

- ▶ Une référence de ce type maintient l'objet vivant
- ▶ Attention aux cycles qui risquent de bloquer la mémoire
- ▶ Mode par défaut

- Weak (relation faible)

- ▶ Une référence de ce type ne maintient pas l'objet vivant
- ▶ Type «conditionnel»
- ▶ Permet de «casser» un cycle
- ▶ Mot clef : weak

- Unowned

- ▶ Weak...
- ▶ Dispose toujours d'une valeur
- ▶ Mot clef : unowned

Comment désallouer?
affecter «nil» à une référence

Retour sur la gestion des liaisons entre objets

54

Différents types de relation

- Strong (relation forte)

- ▶ Une référence de ce type maintient
- ▶ Attention aux cycles qui risquent de
- ▶ Mode par défaut

- Weak (relation faible)

- ▶ Une référence de ce type ne maintient pas l'objet vivant
- ▶ Type «conditionnel»
- ▶ Permet de «casser» un cycle
- ▶ Mot clef : weak

- Unowned

- ▶ Weak...
- ▶ Dispose toujours d'une valeur
- ▶ Mot clef : unowned

Comment désallouer?

affecter «nil» à une référence



Important!!!

Il faut bien utiliser la catégorisation des liaisons pour permettre à ARC de fonctionner...

Exemple 1 — liaisons fortes et cycles

55

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}  
  
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}  
  
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```

Exemple 1 — liaisons fortes et cycles

55

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



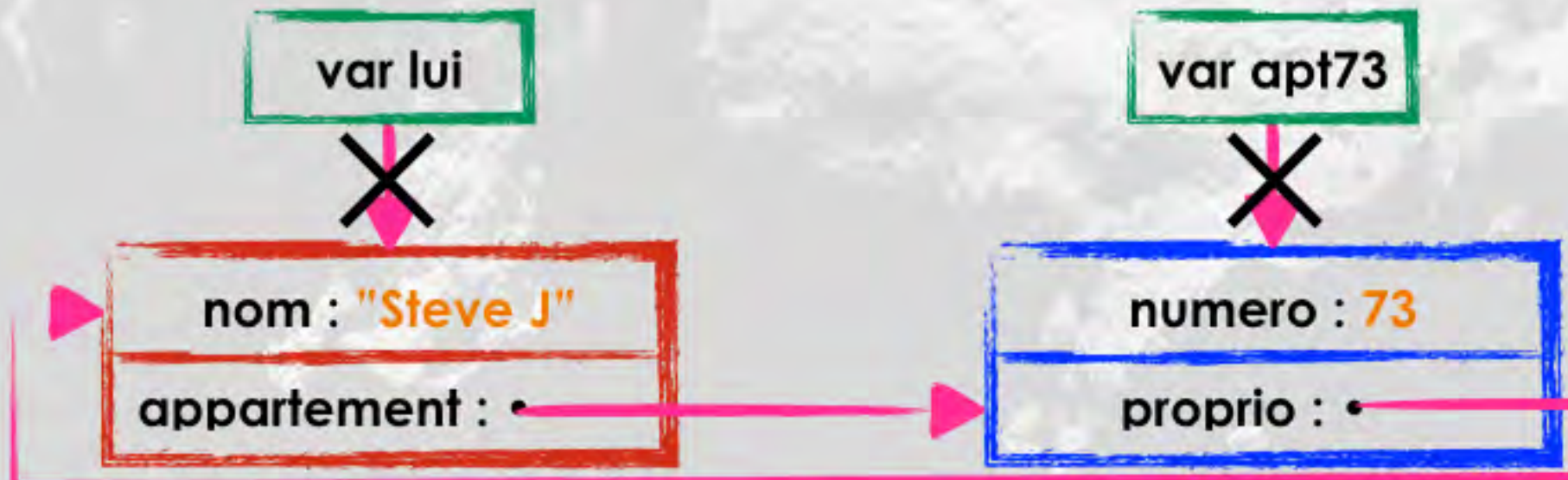
Exemple 1 — liaisons fortes et cycles

55

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



Exemple 1 — liaisons fortes et cycles

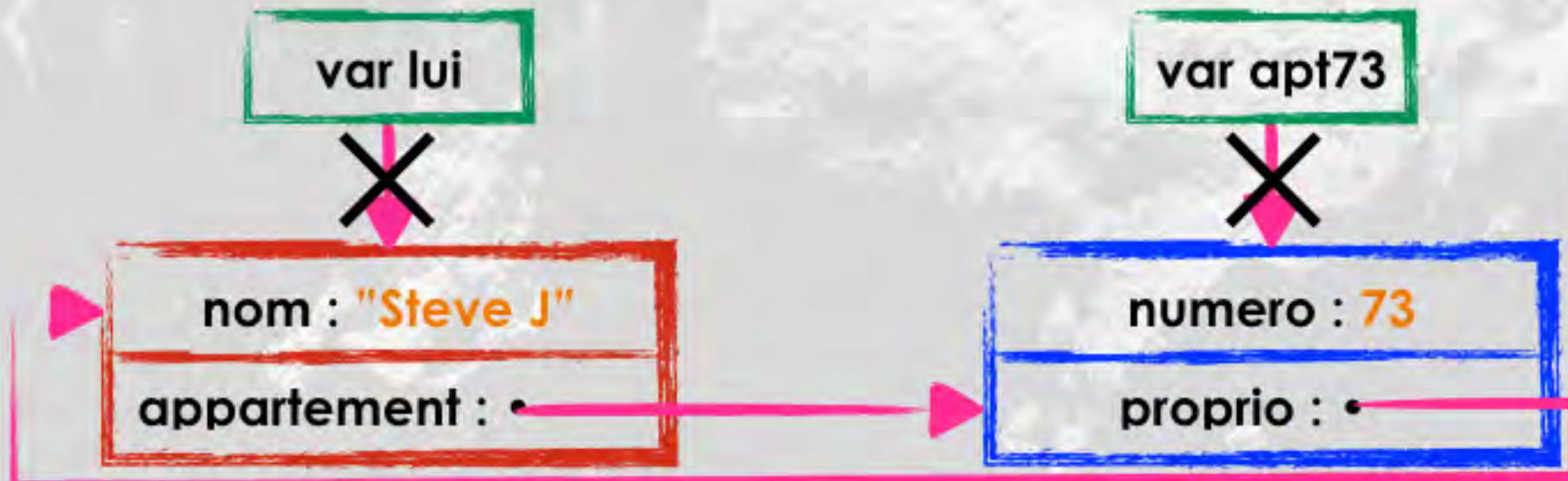
55

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { se  
  var proprio: Person  
  deinit { println("A  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```

Cycle de liaisons fortes
ARC ne sait pas désallouer



Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
    let nom: String  
    init(nom: String) { self.nom = nom }  
    var chezLui: Appartement?  
        deinit { println("\(nom) est «deinitialized»") }  
}  
  
class Appartement {  
    let numero: Int  
    init(num: Int) { self.numero = num }  
    weak var proprio: Personne?  
    deinit { println("Appartement #\(numero) est «deinitialized»") }  
}  
  
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```

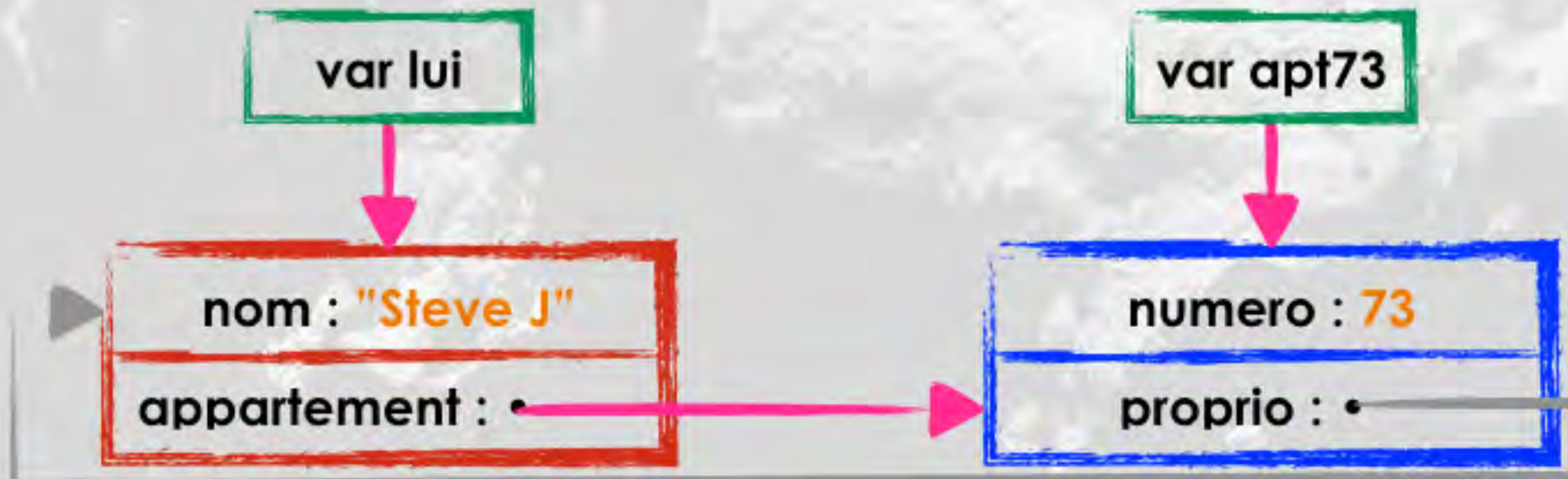
Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



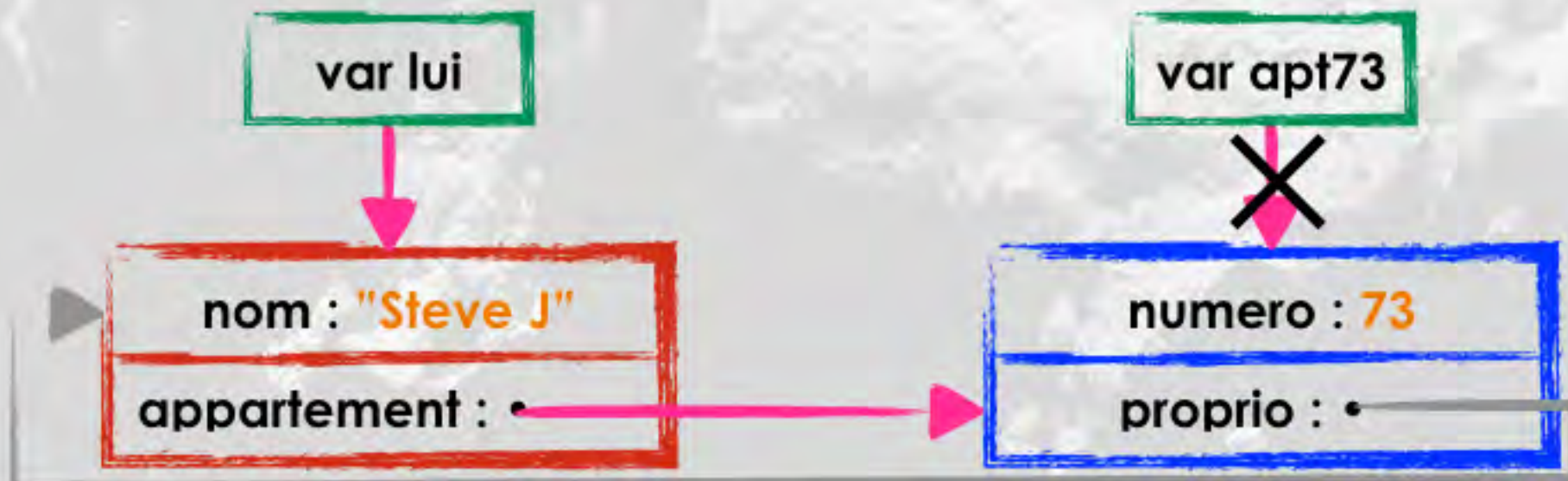
Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



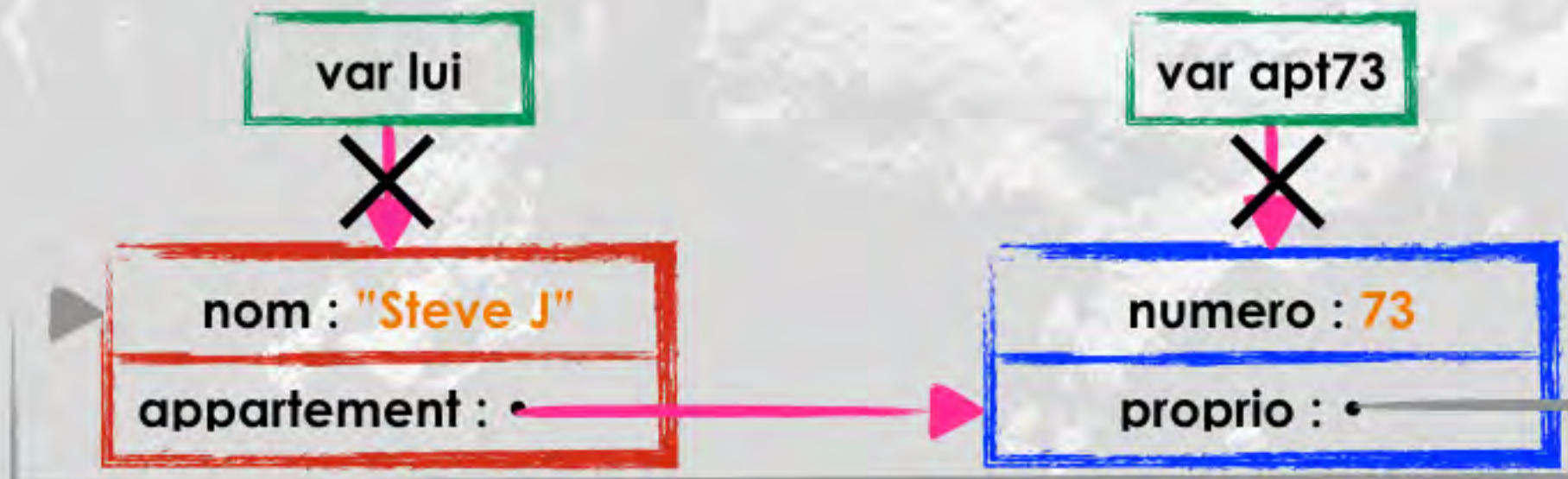
Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



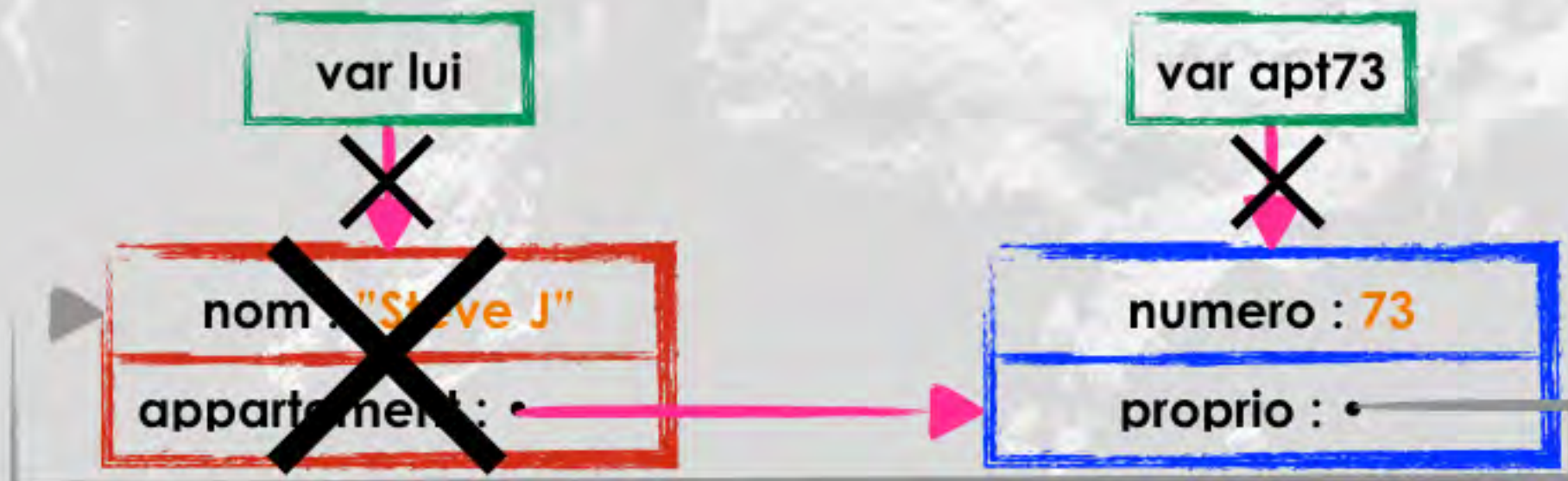
Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



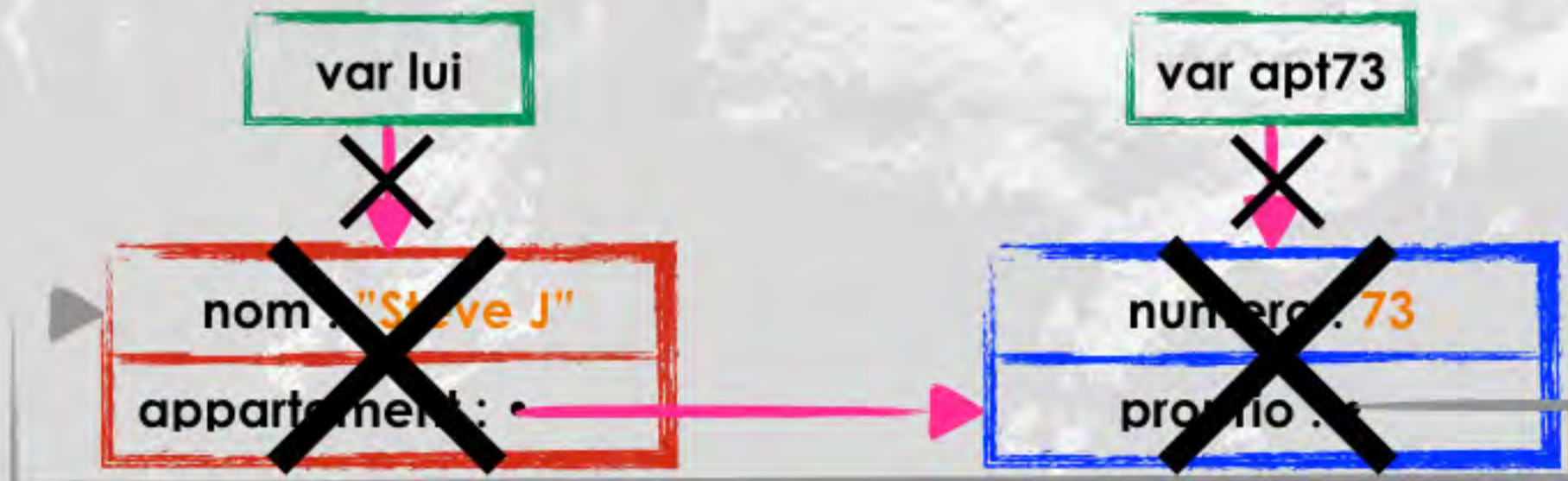
Exemple 2 — casser les cycles (liaison faible)

56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Appartement #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```



Exemple 2 — casser les cycles (liaison faible)

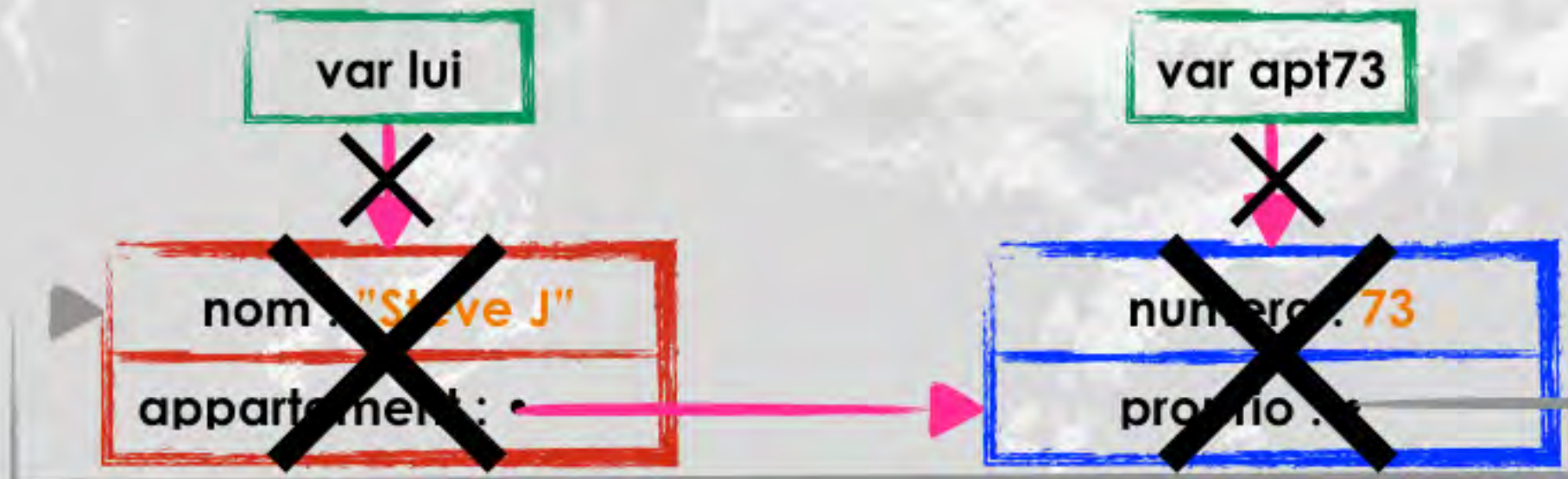
56

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var chezLui: Appartement?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var apt73: Appartement?  
lui = Personne(nom: "Steve J")  
apt73 = Appartement(num: 73)
```

```
class Appartement {  
  let numero: Int  
  init(num: Int) { self.numero = num }  
  weak var proprio: Personne?  
  deinit { println("Apartement «deinitialized»") }  
}
```

Pas de cycle...
ARC sait désallouer



Exemple 3 — «unowned»

57

```
class Personne {
  let nom: String
  init(nom: String) { self.nom = nom }
  var saCarte: CarteDeCredit?
  deinit { println("\(nom) est «deinitialized»") }
}

class CarteDeCredit {
  let numero: Int
  init(num: Int, prop: Personne) { self.numero = num
                                   self.proprio = num }

  unowned let proprio: Personne
  deinit { println("Carte #\(numero) est «deinitialized»") }
}

var lui: Personne?
var carte: CarteDeCredit?
lui = Personne(nom: "Steve J")
carte = CarteDeCredit(num: 73, prop: lui!)
```

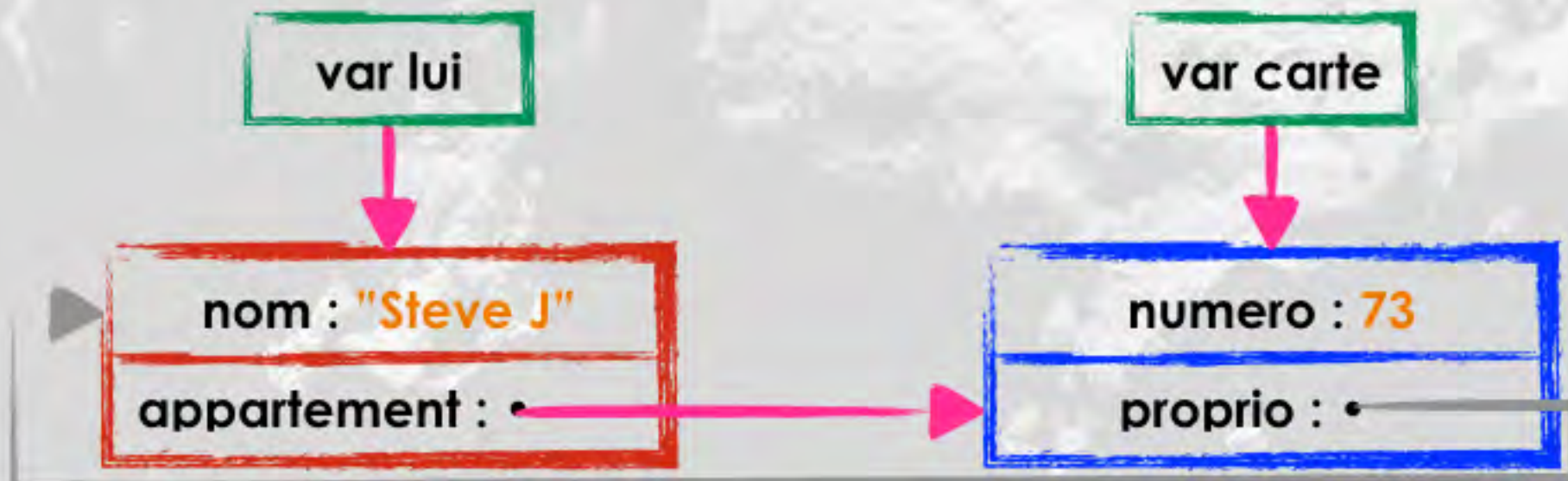
Exemple 3 — «unowned»

57

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var saCarte: CarteDeCredit?  
  deinit { println("\ (nom) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var carte: CarteDeCredit?  
lui = Personne(nom: "Steve J")  
carte = CarteDeCredit(num: 73, prop: lui!)
```

```
class CarteDeCredit {  
  let numero: Int  
  init(num: Int, prop: Personne) { self.numero = num  
                                   self.proprio = num }  
  
  unowned let proprio: Personne  
  deinit { println("Carte #\ (numero) est «deinitialized»") }  
}
```



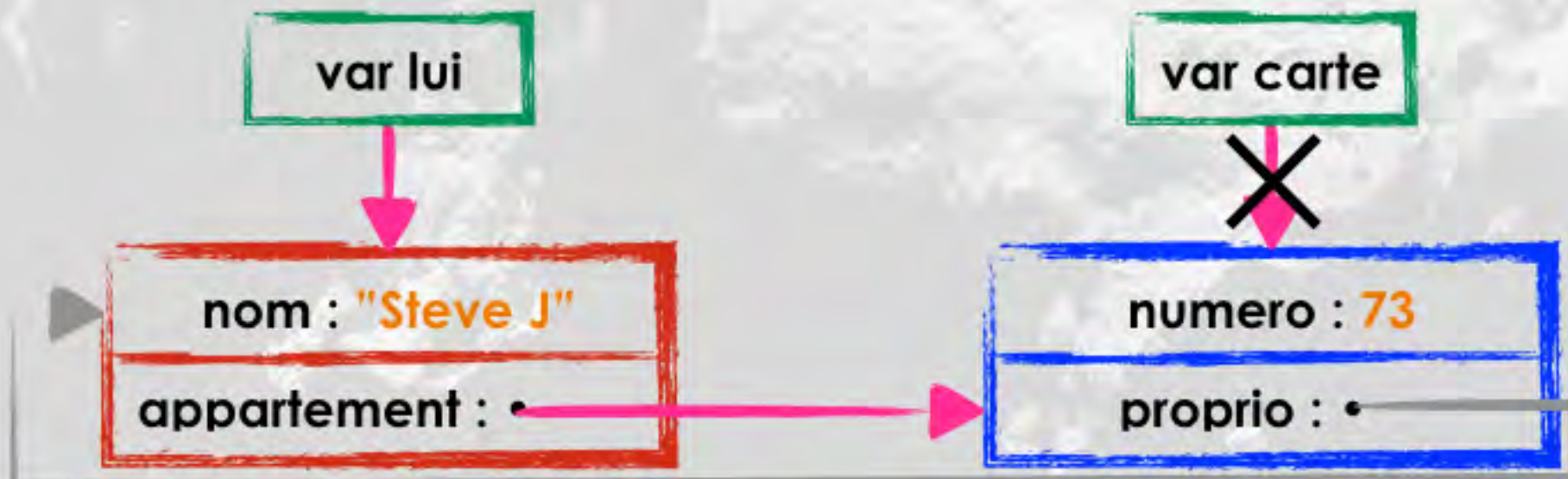
Exemple 3 — «unowned»

57

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var saCarte: CarteDeCredit?  
  deinit { println("\ (nom) est «deinitialized»") }  
}
```

```
class CarteDeCredit {  
  let numero: Int  
  init(num: Int, prop: Personne) { self.numero = num  
                                   self.proprio = num }  
  
  unowned let proprio: Personne  
  deinit { println("Carte #\ (numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var carte: CarteDeCredit?  
lui = Personne(nom: "Steve J")  
carte = CarteDeCredit(num: 73, prop: lui!)
```



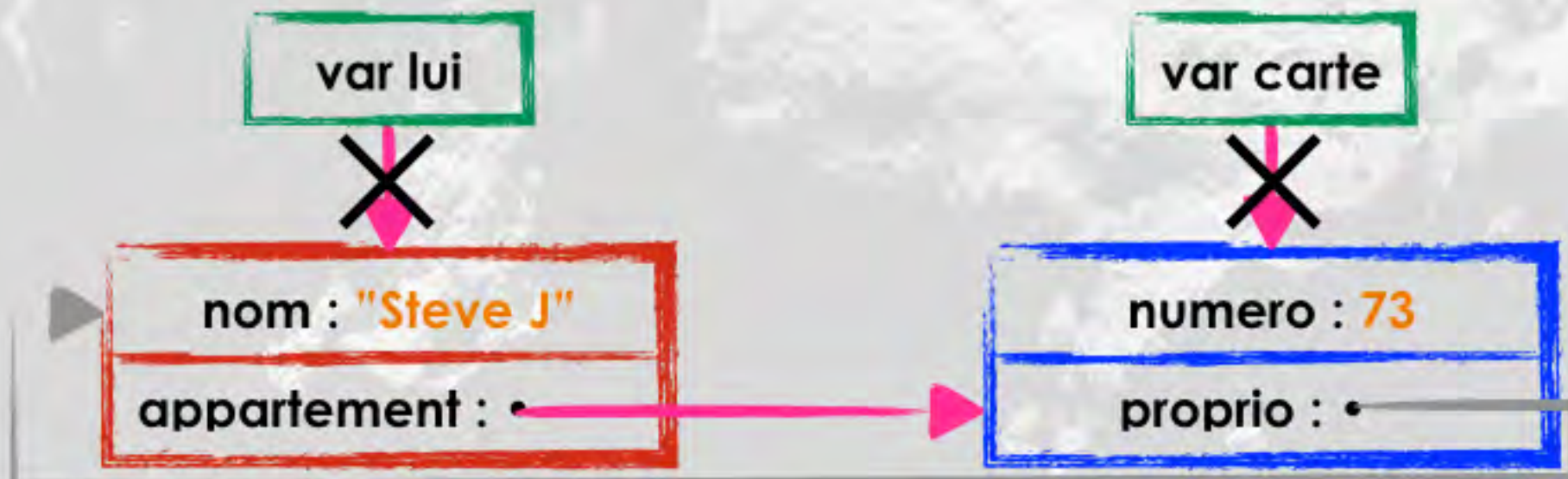
Exemple 3 — «unowned»

57

```
class Personne {  
  let nom: String  
  init(nom: String) { self.nom = nom }  
  var saCarte: CarteDeCredit?  
  deinit { println("\(nom) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var carte: CarteDeCredit?  
lui = Personne(nom: "Steve J")  
carte = CarteDeCredit(num: 73, prop: lui!)
```

```
class CarteDeCredit {  
  let numero: Int  
  init(num: Int, prop: Personne) { self.numero = num  
                                   self.proprio = num }  
  
  unowned let proprio: Personne  
  deinit { println("Carte #\(numero) est «deinitialized»") }  
}
```



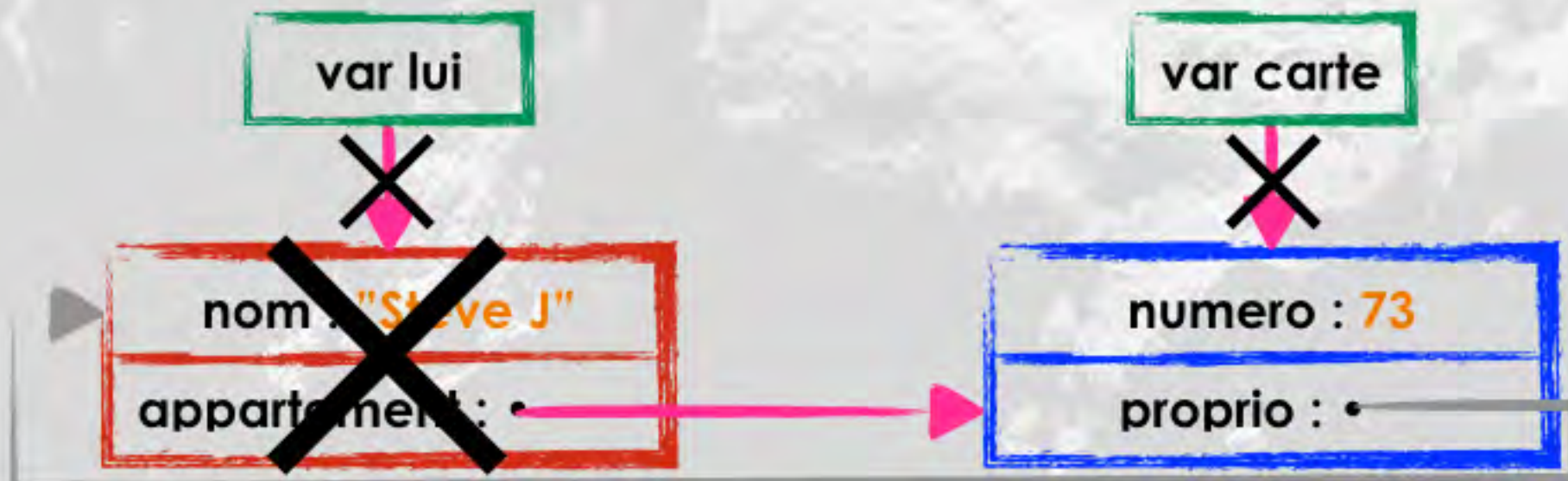
Exemple 3 — «unowned»

57

```
class Personne {  
    let nom: String  
    init(nom: String) { self.nom = nom }  
    var saCarte: CarteDeCredit?  
    deinit { println("\(nom) est «deinitialized»") }  
}
```

```
class CarteDeCredit {  
    let numero: Int  
    init(num: Int, prop: Personne) { self.numero = num  
                                     self.proprio = num }  
  
    unowned let proprio: Personne  
    deinit { println("Carte #\(numero) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var carte: CarteDeCredit?  
lui = Personne(nom: "Steve J")  
carte = CarteDeCredit(num: 73, prop: lui!)
```



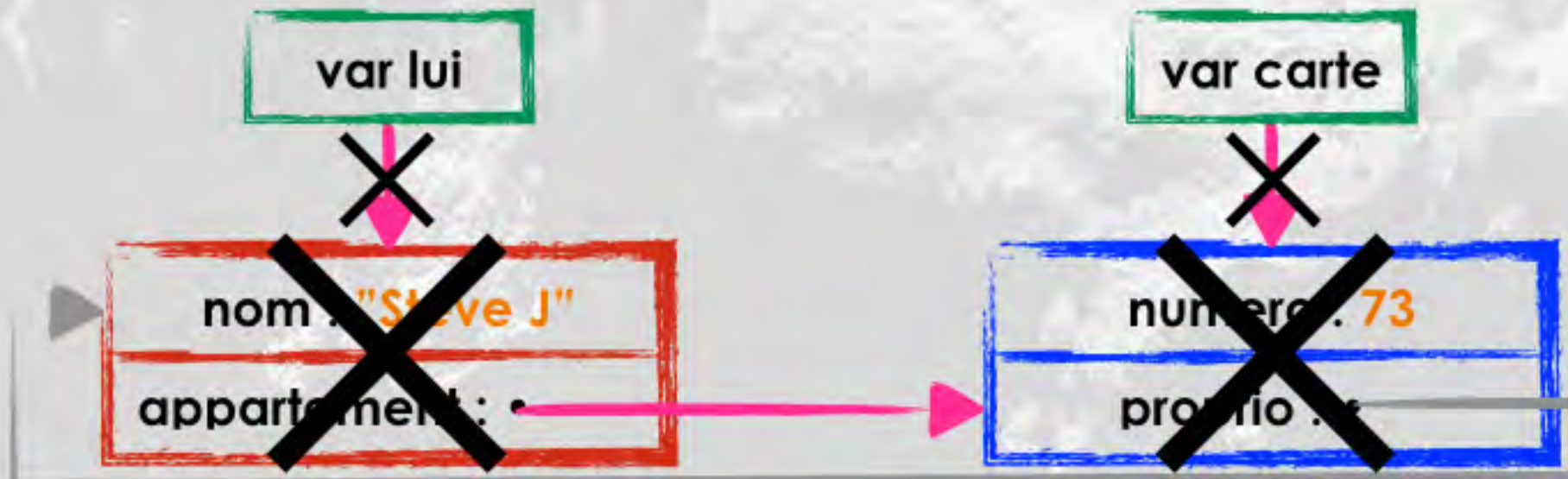
Exemple 3 — «unowned»

57

```
class Personne {  
    let nom: String  
    init(nom: String) { self.nom = nom }  
    var saCarte: CarteDeCredit?  
    deinit { println("\(nom) est «deinitialized»") }  
}
```

```
var lui: Personne?  
var carte: CarteDeCredit?  
lui = Personne(nom: "Steve J")  
carte = CarteDeCredit(num: 73, prop: lui!)
```

```
class CarteDeCredit {  
    let numero: Int  
    init(num: Int, prop: Personne) { self.numero = num  
                                     self.proprio = num }  
  
    unowned let proprio: Personne  
    deinit { println("Carte #\(numero) est «deinitialized»") }  
}
```



Terminaison de l'objet («deinitialization»)

58

📱 Au plus une méthode de terminaison par classe

📱 Objectif

- Faire «ce qu'il faut» avant de terminer l'objet
- Par exemple
 - ▶ Des joueurs au «monopoly»
 - ▶ Chacun dispose d'un portefeuille
 - ▶ Lors de la disparition d'un joueur, le portefeuille doit être restitué à la banque

```
class Player {  
    var laBanque : Banque  
    var dansPortefeuille: Int  
  
    init(valeur: Int, maBanque : Banque) {  
        laBanque = maBanque  
        dansPortefeuille = laBanque.retirer(valeur)  
    }  
    func gagnerArgent(valeur: Int) {  
        dansPortefeuille += laBanque.retirer(valeur)  
    }  
}
```

Terminaison de l'objet («deinitialization»)

58

📱 Au plus une méthode de terminaison par classe

📱 Objectif

- Faire «ce qu'il faut» avant de terminer l'objet
- Par exemple
 - ▶ Des joueurs au «monopoly»
 - ▶ Chacun dispose d'un portefeuille
 - ▶ Lors de la disparition d'un joueur, le portefeuille doit être restitué à la banque

```
deinit {  
    laBanque.deposer(dansPortefeuille)  
}  
}
```

Terminaison de l'objet («deinitialization»)

58

📱 Au plus une méthode de terminaison par classe

📱 Objectif

- Faire «ce

- Par exe

- ▶ Des jou

- ▶ Chacun dispose d'un po

- ▶ Lors de la disparition d'un joueur, le ponere

On assure ainsi la cohérence au niveau de l'application

Cela peut sans doute passer par des désallocations (mettre des références à «nil»)

la banque

```
deinit {  
    laBanque.deposer(dansPortefeuille)  
}  
}
```

En guise de conclusion...

59

📱 **Finally, retain/release/autorelease c'est facile!**

- Il va quand même falloir réfléchir un peu...



📱 **Normal, ARC ne peut fonctionner seul**

- Objective-C se base sur des conventions
 - ▶ **alloc/init/copy, autorelease, etc.**
- Swift s'appuie sur une catégorisation des liaisons

📱 **By the way...**

- Les variables conditionnelles sont initialisées à «nil»

```
var chezLui: Appartement?
```



Petite conclusion sur Swift

Fabrice.Kordon@lip6.fr

Évolution de Swift face à Objective-C

61



Important

- Swift et Objective-C vont encore cohabiter quelques années
 - ▶ Si Swift l'emporte, ce sera long...

Évolution de Swift face à Objective-C

61



Important

- Swift et Objective-C

► Si Swift l'emporte

années

Sep 2014	Sep 2013	Change	Programming Language	Ratings	Change
1	1		C		
2	2		Java	16.721%	-0.25%
3	4	▲	Objective-C	14.140%	-2.01%
4	3	▼	C++	9.935%	+1.37%
5	6	▲	C#	4.674%	-3.99%
6	7	▲	Basic	4.352%	-1.21%
7	5	▼	PHP	3.547%	-1.29%
8	8		Python	3.121%	-3.31%
9	9		JavaScript	2.782%	-0.39%
10	10		Transact-SQL	2.448%	+0.43%
11	11		Visual Basic .NET	1.675%	-0.32%
12	12		Perl	1.532%	-0.31%
13	13		Ruby	1.369%	-0.32%
14	-	▲	Visual Basic	1.281%	-0.10%
15	14	▼	Delphi/Object Pascal	1.272%	+1.27%
16	26	▲	F#	1.157%	+0.26%
17	15	▼	Pascal	0.990%	+0.49%
18	-	▲	Swift	0.893%	+0.01%
19	19		MATLAB	0.852%	+0.85%
				0.818%	+0.16%

Évolution de Swift face à Objective-C

61



Important

- Swift et Objective-C

► Si Swift l'emporte

années

Les deux peuvent interagir!
Au sein d'un même projet

Sep 2014	Sep 2013	Change	Programming Language	Ratings	Change
1	1		C	16.721%	-0.25%
2	2		Java	14.140%	-2.01%
3	4	▲	Objective-C	9.935%	+1.37%
4	3	▼	C++	4.674%	-3.99%
5	6	▲	C#	4.352%	-1.21%
6	7	▲	Basic	3.547%	-1.29%
7	5	▼	PHP	3.121%	-3.31%
8	8		Python	2.782%	-0.39%
9	9		Perl	1.448%	+0.43%
10			JavaScript	1.175%	-0.32%
			Go	0.912%	-0.31%
			Ruby	0.809%	-0.32%
			Visual Basic	1.281%	-0.10%
		▼	Delphi/Object Pascal	1.272%	+1.27%
17	26	▲	F#	1.157%	+0.26%
18	15	▼	Pascal	0.990%	+0.49%
19	-	▲	Swift	0.893%	+0.01%
			MATLAB	0.852%	+0.85%
				0.818%	+0.16%

En guise de conclusion

62

- 📱 Ce n'était qu'un «survol»
- 📱 Il y a encore beaucoup de choses à découvrir
 - Généricité (templates)
 - Pattern matching
 - etc.



En guise de conclusion

62

- 📱 Ce n'était qu'un «survol»
- 📱 Il y a encore beaucoup de choses à découvrir
 - Généricité (templates)
 - Pattern matching
 - etc.



Les vues, principes de base

Fabrice.Kordon@lip6.fr

En guise d'introduction...

64



Vue = élément de base dans iOS

- À commencer par les mécanismes d'interaction

En guise d'introduction...

64

📱 Vue = élément de base dans iOS

- À commencer par les mécanismes d'interaction

📱 Qu'est-ce?

- Un rectangle
 - ▶ Pour y dessiner
 - ▶ Pour insérer d'autres vues
 - ▶ Sensible aux événements
- Organisation hiérarchique
 - ▶ 1 superview
 - ▶ 0+ subviews
- Recouvre ce qui est «dessous»
 - ▶ Opacité...



En guise d'introduction...

64

📱 Vue = élément de base dans iOS

- À commencer par les mécanismes d'interaction

📱 Qu'est-ce?

- Un rectangle
 - ▶ Pour y dessiner
 - ▶ Pour insérer d'autres vues
 - ▶ Sensible aux événements
- Organisation hiérarchique
 - ▶ 1 superview
 - ▶ 0+ subviews
- Recouvre ce qui est
 - ▶ Opacité...



UIView — éléments principaux

65



Personnalisation

- frame
- bounds
- backgroundColor
- hidden
- etc.

position + coordonnées dans la vue parente
origine + taille
couleur du fond (transparent par défaut)
booléen indiquant si la vue est visible



Relations avec les autres vues

- superview
- subviews
- window
- etc.

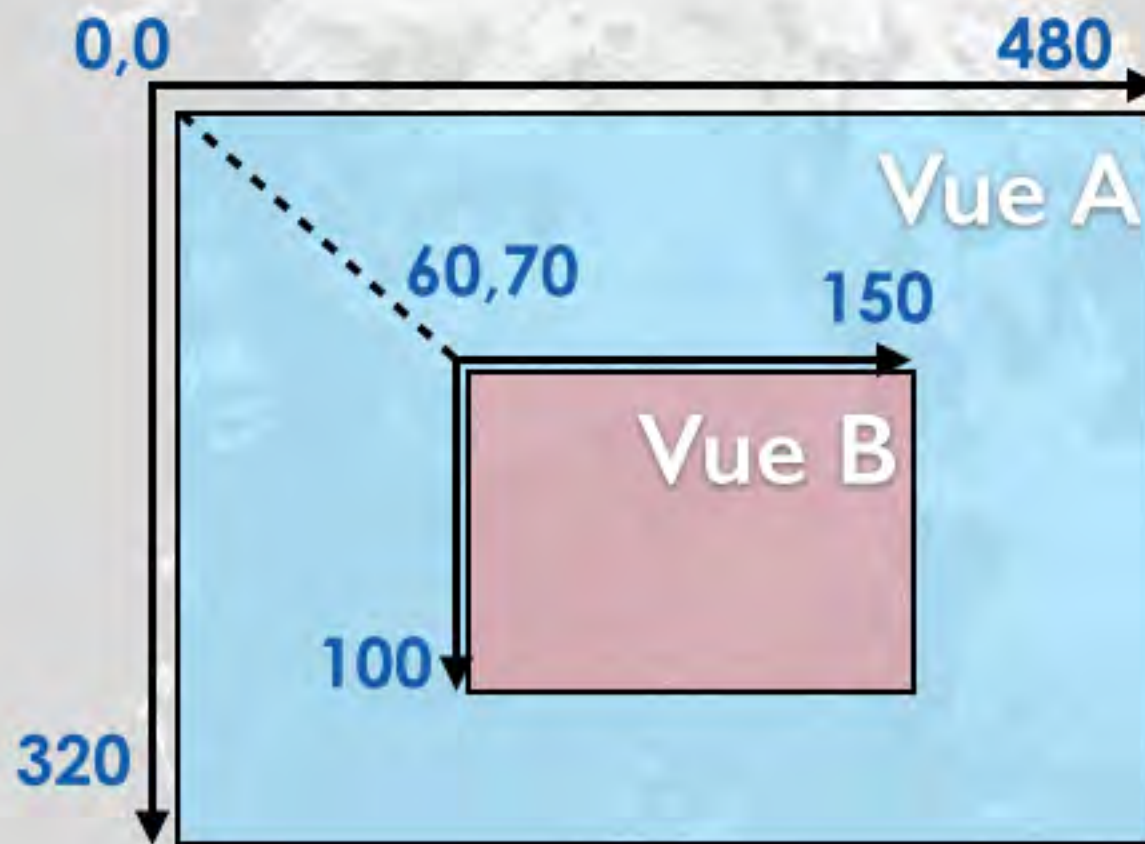
la vue de niveau supérieur
tableau des vues contenues dans la vue
fenêtre contenant la vue (ou nil)



Des coordonnées lié à la vue englobante

66

Par l'exemple...



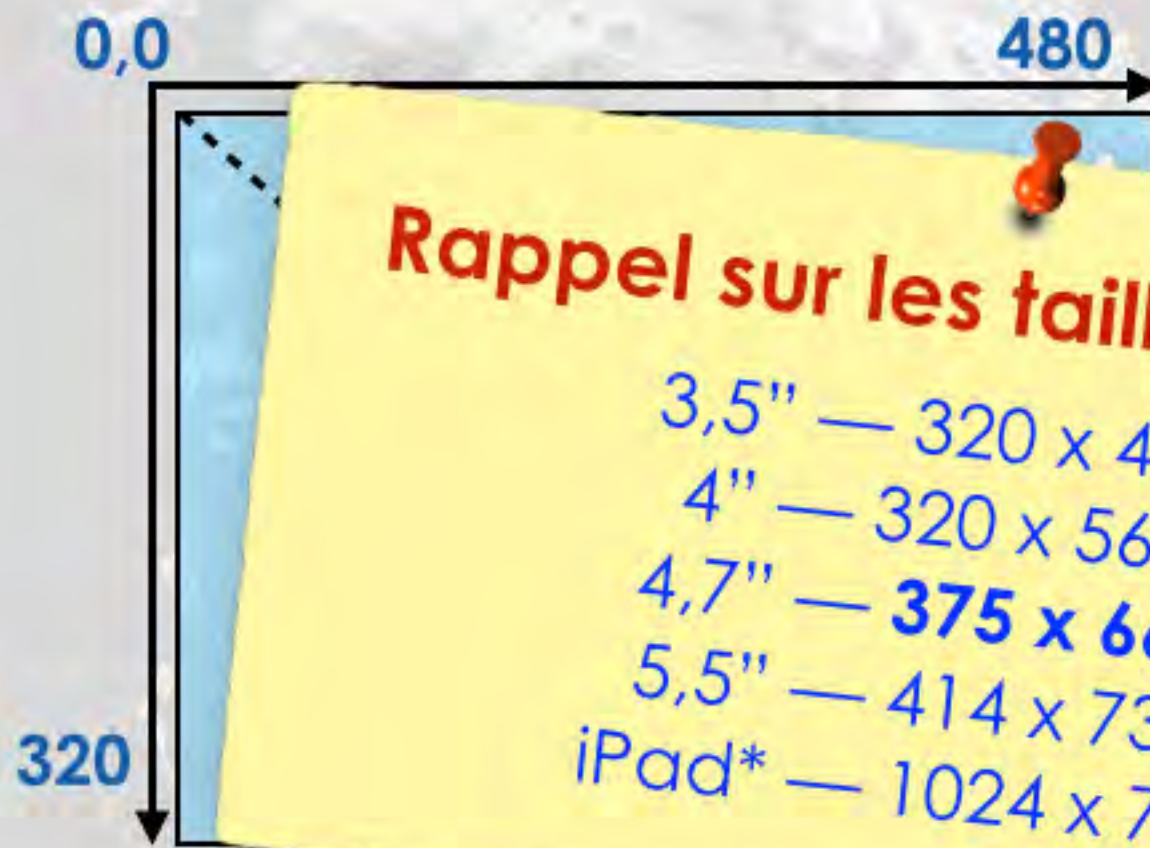
- ▶ **Vue A frame**
origine: 0,0
taille: 480,320
- ▶ **Vue A bounds**
origine: 0,0
taille: 480,320
- ▶ **Vue B frame**
origine: 60,70
taille: 150,100
- ▶ **Vue B bounds**
origine: 0,0
taille: 150,100

- Frame : configuration «à l'extérieur»
- Bounds : configuration «à l'intérieur»

Des coordonnées liées à la vue englobante

66

Par l'exemple...



Rappel sur les tailles d'écrans

3,5" — 320 x 480 pts
4" — 320 x 568 pts
4,7" — **375 x 667** pts
5,5" — 414 x 736 pts
iPad* — 1024 x 768 pts

► **Vue A frame**
origine: 0,0
taille: 480,320

bounds

origine: 0,0
taille: 480,320

frame

origine: 60,70
taille: 150,100

bounds

origine: 0,0
taille: 150,100

- Frame : configuration «à l'extérieur»
- Bounds : configuration «à l'intérieur»

Manipulation des vues

67



Objective-C

- (void)addSubview:(UIView*) view;
- (void)removeFromSuperview;
- (void)insertSubview:(UIView*)view atIndex:(int)index;
- (void)exchangeSubviewAtIndex:(int)index
withSubviewAtIndex:(int)otherIndex;



Swift

```
func addSubview(view: UIView)
func insertSubview(view: UIView,
    aboveSubview siblingSubview: UIView)
func exchangeSubviewAtIndex(index1: Int,
    withSubviewAtIndex index2: Int)
func removeFromSuperview()
```

Manipulation des vues

67

Objective-C

- (void)addSubview:(UIView*) view;
- (void)removeFromSuperview;
- (void)insertSubview:(UIView*) view atIndex:(NSInteger) index;
- (void)exchangeSubviewAtIndex:(NSInteger) index1 withSubviewAtIndex:(NSInteger) index2;

Une vue est propriétaire de ses sous-vues

En objective-C sans ARC, il faut en tenir compte...
... avec des **release!!!**

Swift

```
func addSubview(view: UIView)
func insertSubview(view: UIView,
    aboveSubview siblingSubview: UIView)
func exchangeSubviewAtIndex(index1: Int,
    withSubviewAtIndex index2: Int)
func removeFromSuperview()
```

Construire sa propre classe de type vue

68

- 📱 Créer une classe héritant de UIView
- 📱 Construire le contenu de la classe
- 📱 Éventuellement l'associer à un contrôleur de vue
- 📱 Autres actions importantes
 - Réafficher une partie de la vue
 - (void) drawRect:(CGRect)rect;
 - func drawRect(rect: CGRect)
 - ▶ Utile pour les vues graphiques
 - Forcer le rafraîchissement du contenu
 - (void) setNeedsDisplay;
 - func setNeedsDisplay()

En guise de conclusion...

69

Maintenant il faut «dessiner» dans les vues



Avec UIKit

- Vous êtes condamnés aux rectangles
 - ▶ Mais on peut ruser avec la transparence (cf icônes)
- nombreux «widgets» à votre disposition



Avec CoreGraphics

- Bibliothèque complète de dessin
- Contexte (pour dessiner)
 - ▶ Initialisé par défaut (variable de type CGContextRef à déclarer)
- Formes (notion de «chemin»)
- Couleur
- Polices
- etc.

En guise de conclusion...

69

Maintenant il faut «dessiner» dans les vues



Avec UIKit

- Vous êtes condamnés aux rectangles
 - ▶ Mais on peut ruser avec la transparence (cf icônes)
- nombreux «widge



Avec CoreGra

- Bibliothèque com
- Contexte (pour c
- ▶ Initialisé par défaut (vanilla)
- Formes (notion de «chemin»)
- Couleur
- Polices
- etc.

Autres mécanismes possibles

OpenGL (ES)

SceneKit

Identifier les différents types de terminaux

Fabrice.Kordon@lip6.fr

En guise d'introduction...



Il est important d'identifier les terminaux

- Permet de faire des applications «universelles»
 - ▶ Sauf si l'application n'a de sens que sur un seul type de terminal



L'information est bien sûr connue du système

- Plein d'informations sur le terminal grâce à UIDevice
 - ▶ Nom du terminal,
 - ▶ Type du terminal,
 - ▶ Version du système,
 - ▶ Modèle,
 - ▶ Orientation courante,
 - ▶ Niveau de batterie,
 - ▶ État de la batterie,
 - ▶ Existence et état du détecteur de proximité (iPhone uniquement),
 - ▶ etc.

En guise d'introduction...

71



Il est important d'identifier les terminaux

- Permet de faire des applications «universelles»
 - ▶ Sauf si l'application n'a de sens que sur un seul type de terminal



L'information est bien sûr connue du système

- Plein d'informations sur le terminal grâce à UIDevice
 - ▶ Nom du terminal,
 - ▶ Type du terminal,
 - ▶ Version du système,
 - ▶ Modèle,
 - ▶ Orientation courante,
 - ▶ Niveau de batterie,
 - ▶ État de la batterie,
 - ▶ Existence et état du détecteur de proximité (iPhone uniquement),
 - ▶ etc.



Nos amis!!!
UIDevice & UIScreen

La classe `UIDevice` (premier aperçu)

72

Récupérer une référence...

- `+currentDevice / currentDevice()`

Attributs contenant des informations utiles

- `name`, `systemName`, `systemVersion`, `model`, `localisedModel`, **`userInterfaceIdiom`**, `identifierForVendor`

Orientation du terminal

- `orientation`

État de la batterie

- `batteryLevel`
- `batteryState`

La classe UIDevice (premier aperçu)

72

Récupérer une référence...

- +currentDevice / currentDevice()

Attributs contenant des informations utiles

- name, systemName, systemVersion, model, localisedModel, **userInterfaceIdiom**, identifierForVendor

Orientation du terminal

- orientation

État de la batterie

- batteryLevel
- batteryState

userInterfaceIdiom

Objective-C :

UIUserInterfaceIdiomUnspecified
UIUserInterfaceIdiomPhone
UIUserInterfaceIdiomPad

Swift :

Unspecified, Phone, Pad

📱 Récupérer une référence..

- + mainScreen / mainScreen()
- + screens / screens()

s'il y en a plusieurs

📱 Attributs

- Taille / densité de l'écran
 - ▶ **bounds (pts) / scale (CGFloat)**
- Luminosité
 - ▶ **brightness**
- Bloquer la diminution automatique de luminosité
 - ▶ **wantsSoftwareDimming**

En guise de conclusion...

74

Il est facile de détecter le terminal!!!

- Récupérer une référence
- Consulter le type
- Éventuellement, regarder les caractéristiques de l'écran
 - ▶ 3.5", 4", 4.7" et 5.5"

Vous êtes prêts pour faire programmatiquement des applications universelles...



Vues & contrôleurs de vues

Fabrice.Kordon@lip6.fr

En guise d'introduction...

76

Rappel, dans iOS tout est vue

Les «fonds», les éléments graphiques, les «widgets»

- UILabel, UIButton UISliders, UISegmentedControl, etc
- Vous pouvez même inventer des objets «à la main»

Vous devez donc...

- Les créer
- Associer explicitement des traitements à des événements
- ATTENTION: certains événements ne sont pas supportés par tous les «widget»



Construire une application

77

Jusqu'à maintenant :

- Enrichissement d'un «viewController»
 - ▶ Création des structures de données
 - ▶ Création & agencement des vues
 - ▶ Création des IBAction
 - ▶ Affichage via StoryBoard

Problèmes :

- Pas de séparation entre les données et leur affichage
- Que faire lorsque plusieurs vues doivent se superposer
- Gestion efficace des différents appareils (iPad, iPhone, ...)
 - ▶ Isolation du code (comportement et liaison avec d'autres vues)

Construire une application

77

Jusqu'à maintenant :

- Enrichissement d'un «viewController»
 - ▶ Création des structures de données
 - ▶ Début de l'ajout des vues

MVC, Modèle, Vue, Contrôleur

Mise au point dans les années 1970 (Smalltalk — Xerox)
Lourd mais efficace

Omniprésent en Cocoa (Objective-C & Swift)

Objets pré-construits: les «ViewControllers»
La couche «data» sera vue plus tard (certains «widgets»)

Model View Controller

78

«gabarit de conception»



Le modèle

- Traitement des données + gestion de leur intégrité
- Offre des méthodes d'accès et de manipulation (setter/getter)



La vue

- Interaction avec l'utilisateur (entrées & sorties)



Le contrôleur

- Gestion des événements
 - ▶ Mise à jour de la vue
 - ▶ Synchronisation entre le modèle et la vue

Model View Controller

«gabarit de conception»



78



Le modèle

- Traitement des données + gestion de leur intégrité
- Offre des méthodes d'accès et de manipulation (setter/getter)



La vue

- Interaction avec l'utilisateur (entrées & sorties)



Le contrôleur

- Gestion des événements
 - ▶ Mise à jour de la vue
 - ▶ Synchronisation entre le modèle et la vue

Model View Controller

Avantage

Architecture claire!!!

« de conception »



- Traitement des données + gestion de leur intégrité
- Offre des méthodes d'accès et de manipulation (setter/getter)

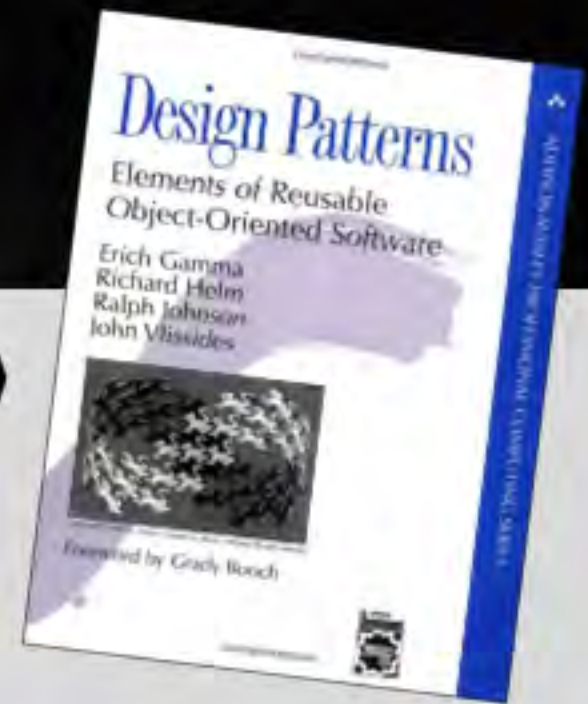
La vue

- Interaction avec l'utilisateur (entrées & sorties)

Le contrôleur

- Gestion des événements
 - ▶ Mise à jour de la vue
 - ▶ Synchronisation entre le modèle et la vue

Model View Controller



78

Avantage

Architecture claire!!!

« de conception »

- Traitement des données + gestion de leur intégrité
- Offre des méthodes d'accès et de manipulation (setter/getter)

La vue

- Interaction avec l'utilisateur

Le contrôleur

- Gestion des événements

Concrètement!!!

Contrôleur = `UIViewController`

Vue = `UIView`

Modèle = une/des classes gérant les données

- ▶ Mise à jour de la vue
- ▶ Synchronisation entre le modèle et la vue

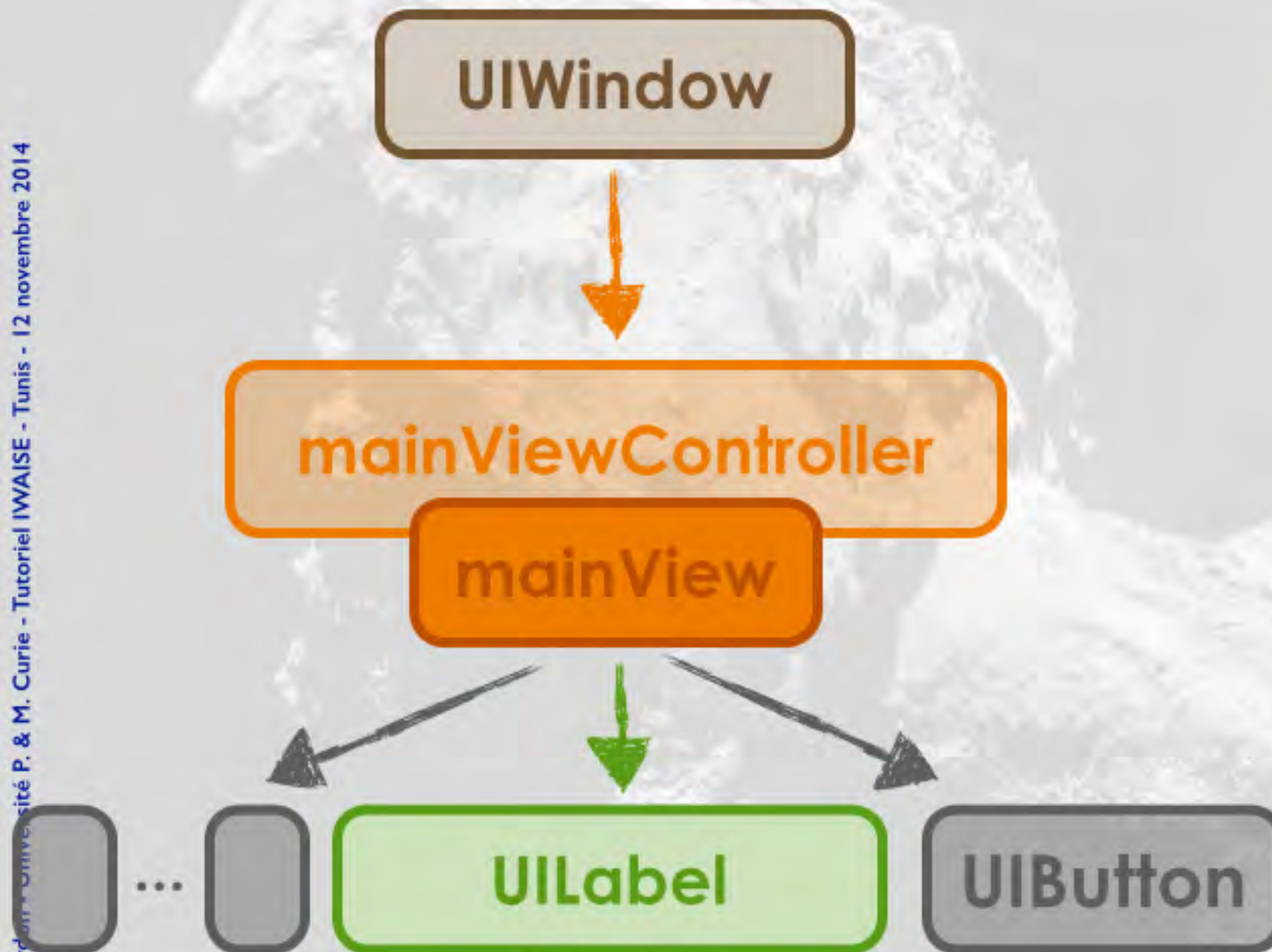
Structuration d'une vue

79



Structuration d'une vue

79



Le cycle de vie d'un ViewController

80

- 📱 **Init ou initWithFrame (ou initWithNibName)**
- 📱 **ViewDidLoad (initialisation & création)**
 - Permet de construire le contenu (aussi dans loadView)
- 📱 **D'autres méthodes possibles**
 - isViewLoaded, viewWillAppear, viewDidAppear, viewWillDisappear,...
 - ▶ RTFM
- 📱 **didReceiveMemoryWarning**
 - Alerte en cas de besoin de mémoire
 - ▶ Disposer de ce qui peut l'être
- 📱 **Gestion de la rotation**
 - Présenté plus loin

Créer une vue dans l'application...

81



Créer une classe qui hérite de UIView

- Peut supporter le protocole UIScrollViewDelegate



Compléter loadView ou viewDidLoad dans le contrôleur de vue

- Allocation/initialisation de la vue
- Création de la «filiation» (à la vue racine associée)

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // Do any additional setup after loading the view  
    UIScreen *ecran = [UIScreen mainScreen];  
    CGRect rect = [ecran bounds];  
    MaVue *v = [[MaVue alloc] initWithFrame:rect];  
    [self setView:v];  
    [v release];  
}
```

Créer une vue dans l'application...

81



Créer une classe qui hérite de UIView

- Peut supporter le protocole UIScrollViewDelegate



Compléter loadView ou viewDidLoad dans le contrôleur de vue

- Allocation/initialisation de la vue
- Création de la «filiation» (à la vue racine associée)

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    let ekran = UIScreen.mainScreen()  
    let rect = ekran.bounds  
    let v = MaVue(frame: rect)  
    self.view = v  
}
```

En guise de conclusion...

82

Finalement, c'est plus simple que Storyboard...

- Il faut juste gérer les coordonnées

Conseil avisé...

- Prenez le temps de «dessiner»
- Pensez en position relative...



Scoop !

- Il existe un système de gestion de contraintes, **AutoLayouts**



UPMC

SORBONNE UNIVERSITÉS

«MaVue», construction programmatique en Swift

Fabrice.Kordon@lip6.fr

En guise d'introduction

84

Un exemple sans StoryBoard (juste pour voir)



Construction programmatique d'une vue

- En Swift
- Gestion explicite des coordonnées (en points)
- Gestion de l'interaction entre le contrôleur et sa vue associée
- Modèle réduit ici à son strict minimum

▶ Car trop simple 

En avant!

L'application en elle-même...

85



L'application en elle-même...

85



ViewController.swift

86

```
//  
// ViewController.swift  
// MaVue-swift  
//  
// Created by Fabrice Kordon on 02/10/2014.  
// Copyright (c) 2014 Fabrice Kordon. All rights reserved.  
//  
  
import UIKit  
  
class ViewController: UIViewController {  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        // Do any additional setup after loading the view,  
  
    }  
}
```

ViewController.swift

86

```
//  
// ViewController.swift  
// MaVue-swift  
//  
// Created by Fabrice Kordon on 02/10/2014.  
// Copyright (c) 2014 Fabrice Kordon. All rights reserved.  
//  
  
import UIKit  
  
class ViewController: UIViewController {  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        // Do any additional setup after loading the view,  
        let ecran = UIScreen.mainScreen()  
        let rect = ecran.bounds  
        let v = MaVue(frame: rect)  
        self.view = v  
    }  
}
```

MaVue.swift — par défaut

87

```
//  
// MaVue.swift  
// bidon2  
//  
// Created by Fabrice Kordon on 03/10/2014.  
// Copyright (c) 2014 Fabrice Kordon. All rights reserved.  
//  
  
import UIKit  
  
class MaVue: UIView {  
  
    /*  
    // Only override drawRect: if you perform custom drawing.  
    // An empty implementation adversely affects performance during animation.  
    override func drawRect(rect: CGRect)  
    {  
        // Drawing code  
    }  
    */  
}
```

**Ce qui est proposé
par défaut**

MaVue.swift — par défaut

87

```
//  
// MaVue.swift  
// bidon2  
//  
// Created by Fabrice Kordon on 03/10/2014.  
// Copyright (c) 2014 Fabrice Kordon. All rights reserved.  
//
```

```
import UIKit
```

```
class MaVue: UIView {
```

```
    /*  
    // Only override drawRect: if you perform custom drawing.  
    // An empty implementation does nothing but may affect performance during animation.  
    override func drawRect(rect: CGRect)  
    {  
        // Drawing code  
    }  
    */  
}
```

Ce qu'il faut faire ici...

- 1) créer un initialiseur
- 2) décommenter drawRect
- 3) remplir drawRect

MaVue.swift — attributs

```
private let terminal = UIDevice.currentDevice()
private let screen = UIScreen.mainScreen()

private let terminfo = UILabel()
private let model = UILabel()
private let orientation = UILabel()

private var sommetLabels = 0
private var increment = 0
```

MaVue.swift — attributs

88

```
private let terminal = UIDevice.currentDevice()  
private let screen = UIScreen.mainScreen()  
  
private let terminfo = UILabel()  
private let model = UILabel()  
private let orientation = UILabel()  
  
private var sommetLabels = 0  
private var increment = 0
```

Différence avec Objective-C
Initialisation des classes
à la déclaration

MaVue.swift — init

89

```
override init (frame: CGRect) {
```

MaVue.swift — init

89

```
override init (frame: CGRect) {
```

Qu'est-ce que frame?

La taille de la vue
(ici l'écran, cf viewDidLoad)

MaVue.swift — init

89

```
override init (frame: CGRect) {  
    terminfo.textAlignment = .Center  
    model.textAlignment = .Center  
    orientation.textAlignment = .Center  
    if terminal.userInterfaceIdiom == .Phone && screen.scale == 3.0 {  
        terminfo.text = "iPhone 6+ (\(terminal.systemName), \(terminal.systemVersion))"  
        increment = 50  
    } else if terminal.userInterfaceIdiom == .Phone {  
        terminfo.text = "iPhone/iPod (\(terminal.systemName), \(terminal.systemVersion))"  
        increment = 40  
    } else {  
        terminfo.text = "iPad (\(terminal.systemName), \(terminal.systemVersion))"  
        increment = 100  
    }  
    model.text = terminal.model  
}
```

}

MaVue.swift — init

```
override init (frame: CGRect) {
    terminfo.textAlignment = .Center
    model.textAlignment = .Center
    orientation.textAlignment = .Center
    if terminal.userInterfaceIdiom == .Phone && screen.scale == 3.0 {
        terminfo.text = "iPhone 6+ (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 50
    } else if terminal.userInterfaceIdiom == .Phone {
        terminfo.text = "iPhone/iPod (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 40
    } else {
        terminfo.text = "iPad (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 100
    }
    model.text = terminal.model

    super.init(frame: frame)
}
```

MaVue.swift — init

89

```
override init (frame: CGRect) {
    terminfo.textAlignment = .Center
    model.textAlignment = .Center
    orientation.textAlignment = .Center
    if terminal.userInterfaceIdiom == .Phone && screen.scale == 3.0 {
        terminfo.text = "iPhone 6+ (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 50
    } else if terminal.userInterfaceIdiom == .Phone {
        terminfo.text = "iPhone/iPod (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 40
    } else {
        terminfo.text = "iPad (\(terminal.systemName), \(terminal.systemVersion))"
        increment = 100
    }
    model.text = terminal.model

    super.init(frame: frame)
    // Après car les attributs suivants appartiennent au super
    self.backgroundColor = UIColor.whiteColor()

    self.addSubview(terminfo)
    self.addSubview(model)
    self.addSubview(orientation)
}
```

MaVue.swift — drawRect

90

```
override func drawRect(rect: CGRect) {
```

MaVue.swift — drawRect

90

```
override func drawRect(rect: CGRect) {
```

Qu'est-ce que rect?

La zone à (ré)afficher
Appel géré par l'environnement

MaVue.swift — drawRect

90

```
override func drawRect(rect: CGRect) {  
    // Drawing code  
  
    if terminal.orientation == UIDeviceOrientation.Portrait ||  
       terminal.orientation == UIDeviceOrientation.PortraitUpsideDown {  
        orientation.text = "Orientation portrait"  
        sommetLabels = 100  
    } else if terminal.orientation == UIDeviceOrientation.LandscapeLeft ||  
               terminal.orientation == UIDeviceOrientation.LandscapeRight {  
        orientation.text = "Orientation paysage"  
        sommetLabels = 50  
    } else {  
        orientation.text = "terminal à plat"  
    }  
}
```

MaVue.swift — drawRect

90

```
override func drawRect(rect: CGRect) {  
    // Drawing code  
  
    if terminal.orientation == UIDeviceOrientation.Portrait ||  
       terminal.orientation == UIDeviceOrientation.PortraitUpsideDown {  
        orientation.text = "Orientation portrait"  
        sommetLabels = 100  
    } else if terminal.orientation == UIDeviceOrientation.LandscapeLeft ||  
              terminal.orientation == UIDeviceOrientation.LandscapeRight {  
        orientation.text = "Orientation paysage"  
        sommetLabels = 50  
    } else {  
        orientation.text = "terminal à plat"  
    }  
  
    terminfo.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),  
                                 CGFloat(sommetLabels), 300.0, 21.0)  
    model.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),  
                             CGFloat(sommetLabels + increment), 300.0, 21.0)  
    orientation.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),  
                                   CGFloat(sommetLabels + 2 * increment), 300.0, 21.0)
```

MaVue.swift — drawRect

90

```
override func drawRect(rect: CGRect) {
```

```
    // Drawing code
```

```
    if terminal.orientation == UIDeviceOrientation.Portrait ||
       terminal.orientation == UIDeviceOrientation.PortraitUpsideDown {
        orientation.text = "Orientation portrait"
        sommetLabels = 100
    } else if terminal.orientation == UIDeviceOrientation.LandscapeLeft ||
       terminal.orientation == UIDeviceOrientation.LandscapeRight {
        orientation.text = "Orientation paysage"
        sommetLabels = 50
    } else {
        orientation.text = "terminal à plat"
    }
}
```

```
    terminfo.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),
                                   CGFloat(sommetLabels), 300.0, 21.0)
    model.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),
                              CGFloat(sommetLabels + increment), 300.0, 21.0)
    orientation.frame = CGRectMake(CGFloat(rect.size.width / 2.0 - 150.0),
                                    CGFloat(sommetLabels + 2 * increment), 300.0, 21.0)
```

```
    var ctx = UIGraphicsGetCurrentContext()
```

```
    let table = [(0.9, 70.0), (0.8, 60.0), (0.7, 50.0), (0.6, 40.0), (0.5, 30.0),
                 (0.3, 20.0), (0.0, 10.0),]
```

```
    for (vert, rayon) in table {
```

```
        UIColor(red: 1.0, green: CGFloat(vert), blue: 0.0, alpha: 1.0).setFill()
```

```
        CGContextAddArc(ctx, CGFloat(rect.size.width / 2.0),
```

```
                        CGFloat(rect.size.height / 2.0 + 70.0), CGFloat(rayon),
```

```
                        0.0, CGFloat(M_PI * 2), 0)
```

```
        CGContextFillPath(ctx);
```

```
    }
```

```
}
```

En guise de conclusion...

91

📱 **Enfin, ce n'est pas si difficile non?**

- Pas plus qu'en Objective-C



📱 **Il faut juste réfléchir un peu...**



📱 **On n'est pas des «mangeur de quiche» ;-)**



«NuancierRVB»

Fabrice.Kordon@lip6.fr

En guise d'introduction...

93

Objectifs de l'exercice

- Regarder certains mécanismes pour l'interactivité sous iOS
- Les traiter à la fois en «outlets» et en «actions»
 - ▶ **Préserver la cohérence de l'interface**
- Traiter une interface complète avec Storyboard
 - ▶ **Gérer de manière différenciée le terminaux et les orientations**
- Écrire de l'objective C (dans les méthodes)

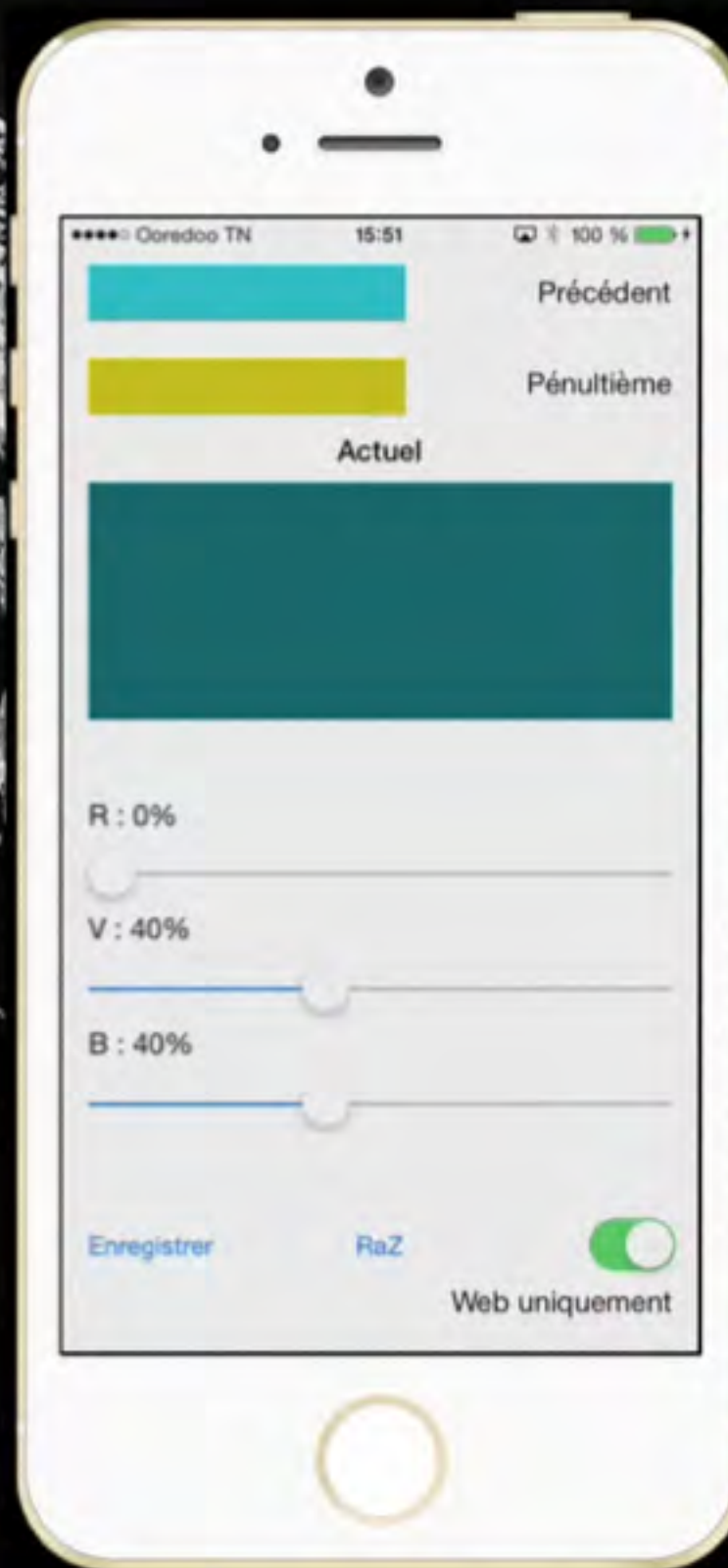
Une application sérieuse

- Il en existe d'assez voisines dans l'AppStore...



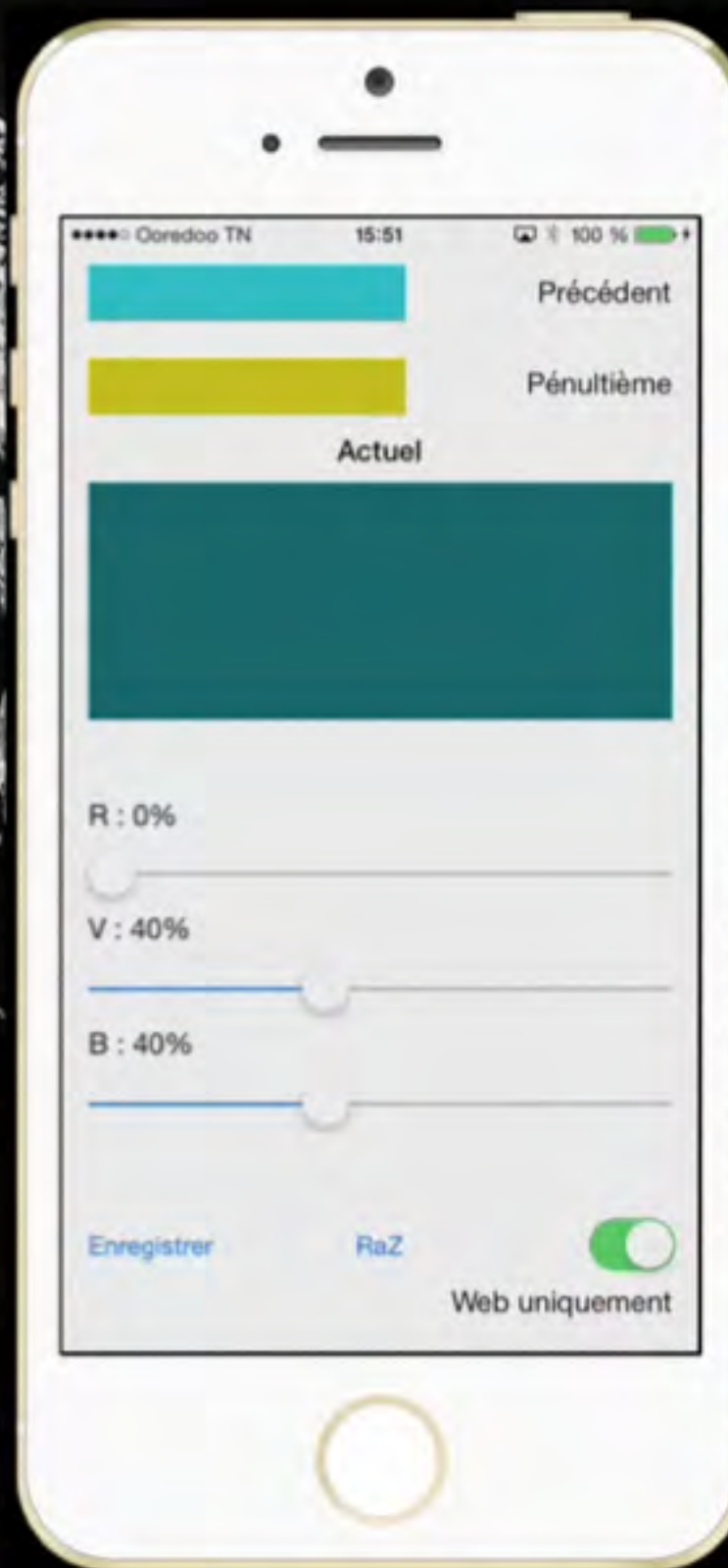
Petite démonstration

94



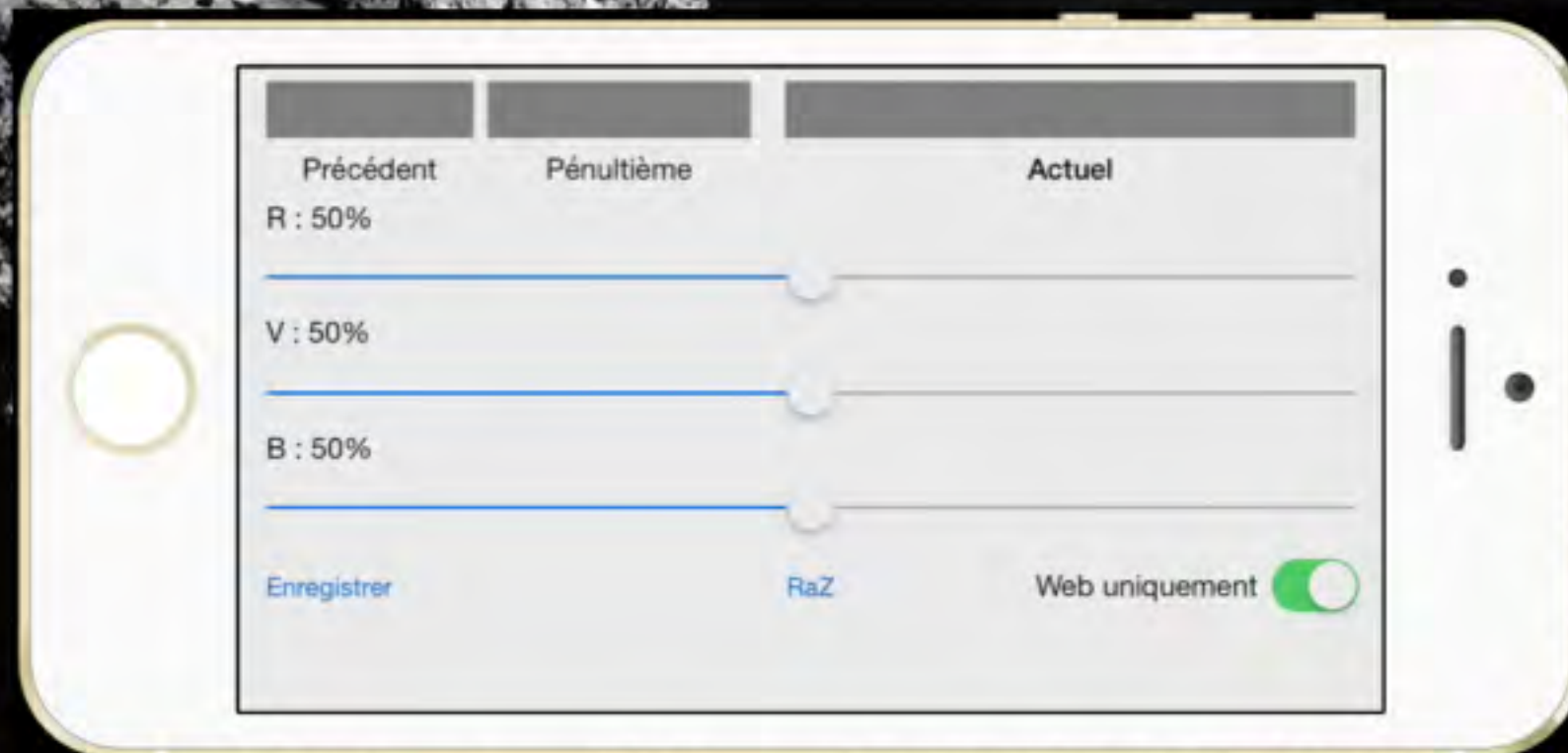
Petite démonstration

94



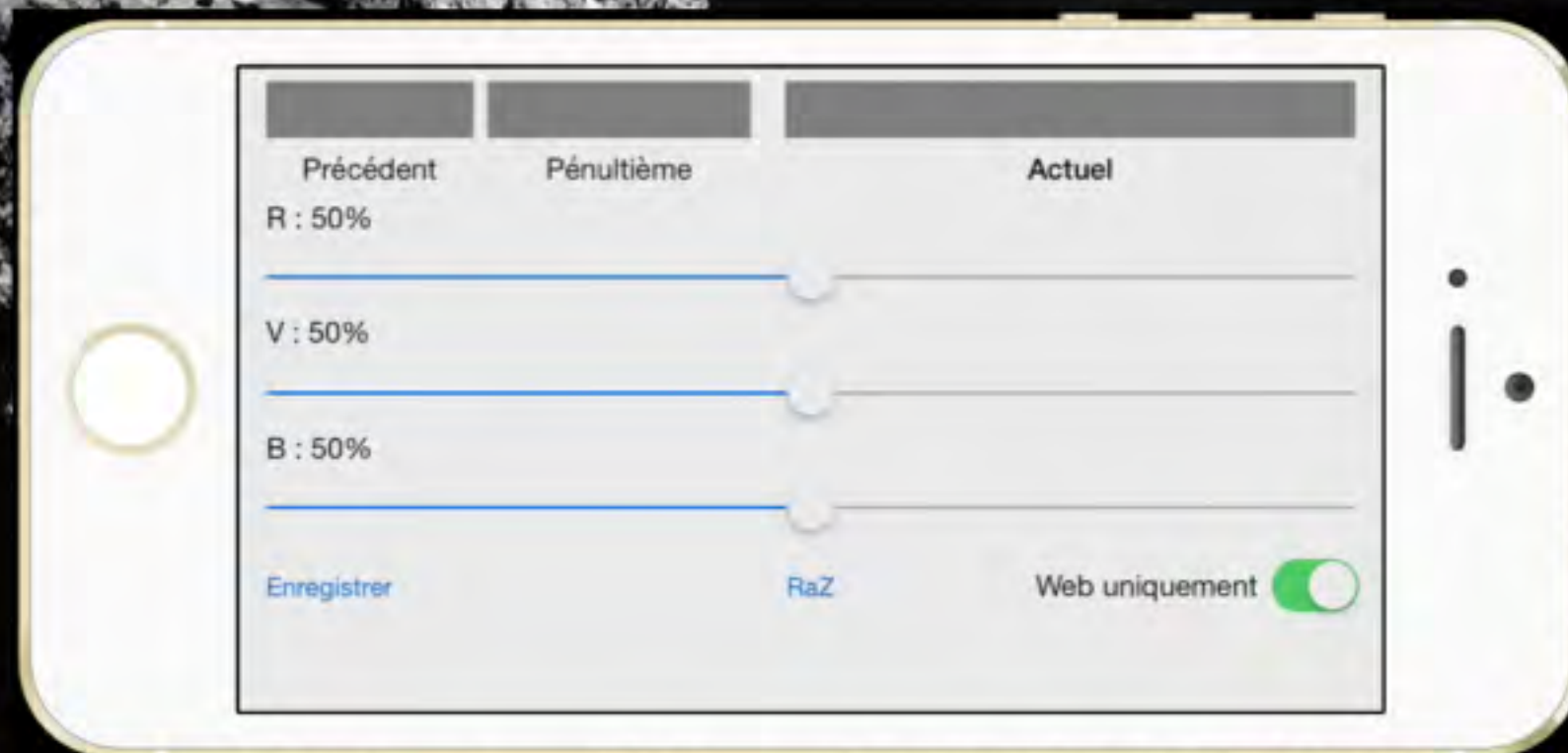
Et en mode horizontal?

95



Et en mode horizontal?

95



Quelques éléments

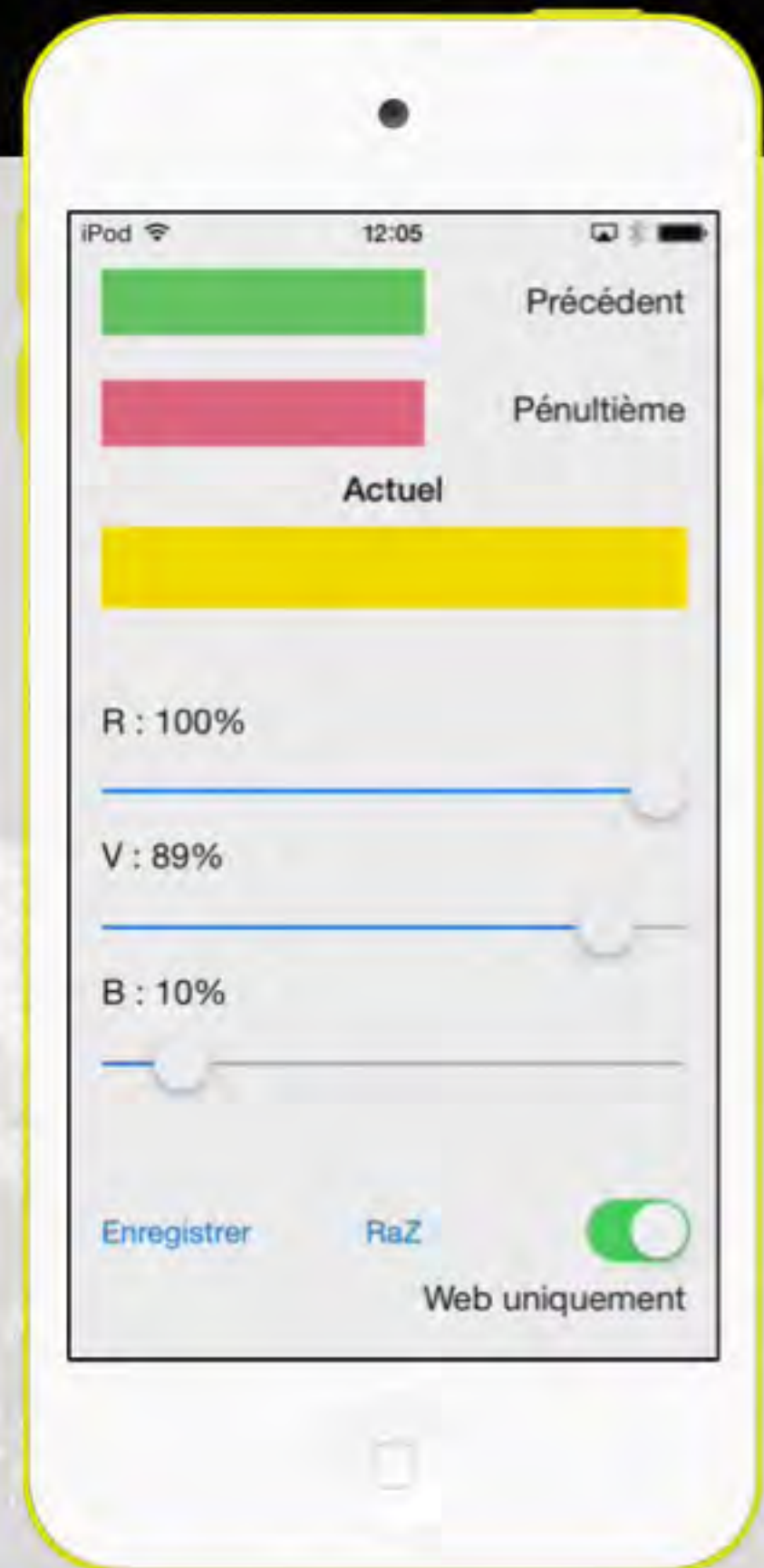
Les mécanismes que vous utiliserez

- UIButton + UILabel
 - ▶ «Boutons» étranges (fond coloré, labels différenciés)
- UISwitch
- UISlider
- UIView
 - ▶ Pour afficher la «couleur principale»

Plus du code...

- Formatage dans NSString
- Gestion des couleurs

Bref, de quoi...



En guise de conclusion...

97

Amusez-vous bien!!!



Vous pouvez enrichir l'exercice si cela vous chante!

«Quizz»

Fabrice.Kordon@lip6.fr

En guise d'introduction...

Objectifs de l'exercice

- Regarder les principaux mécanismes pour l'interactivité sous iOS
- Prendre de l'expérience sur Storyboard
- Écrire du swift
 - ▶ Dans les méthodes à remplir sur le «ViewController»
 - ▶ Créer et utiliser une classe en swift

Une application sérieuse

- Il en existe des dérivés sur l'Appstore



Petite démonstration

100



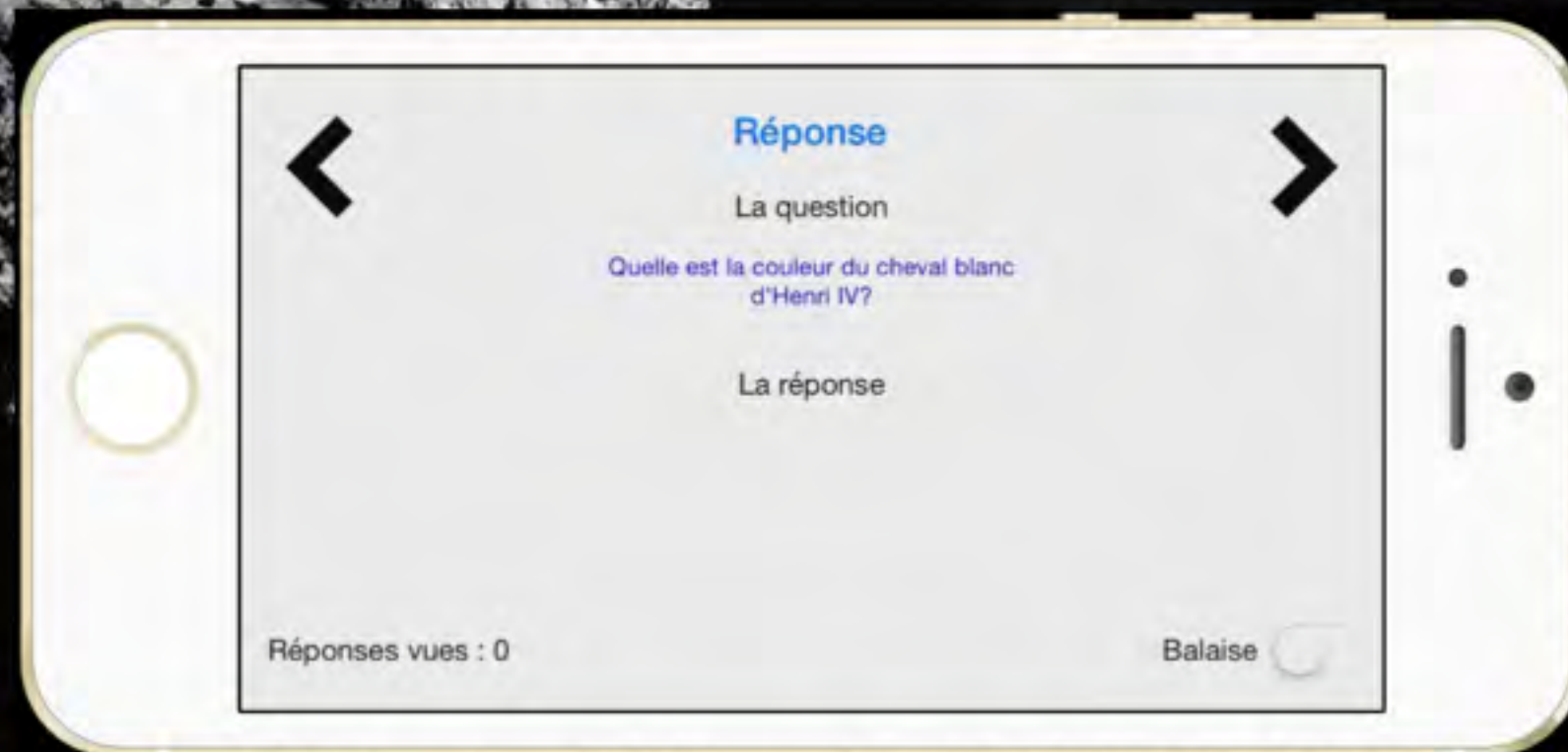
Petite démonstration

100



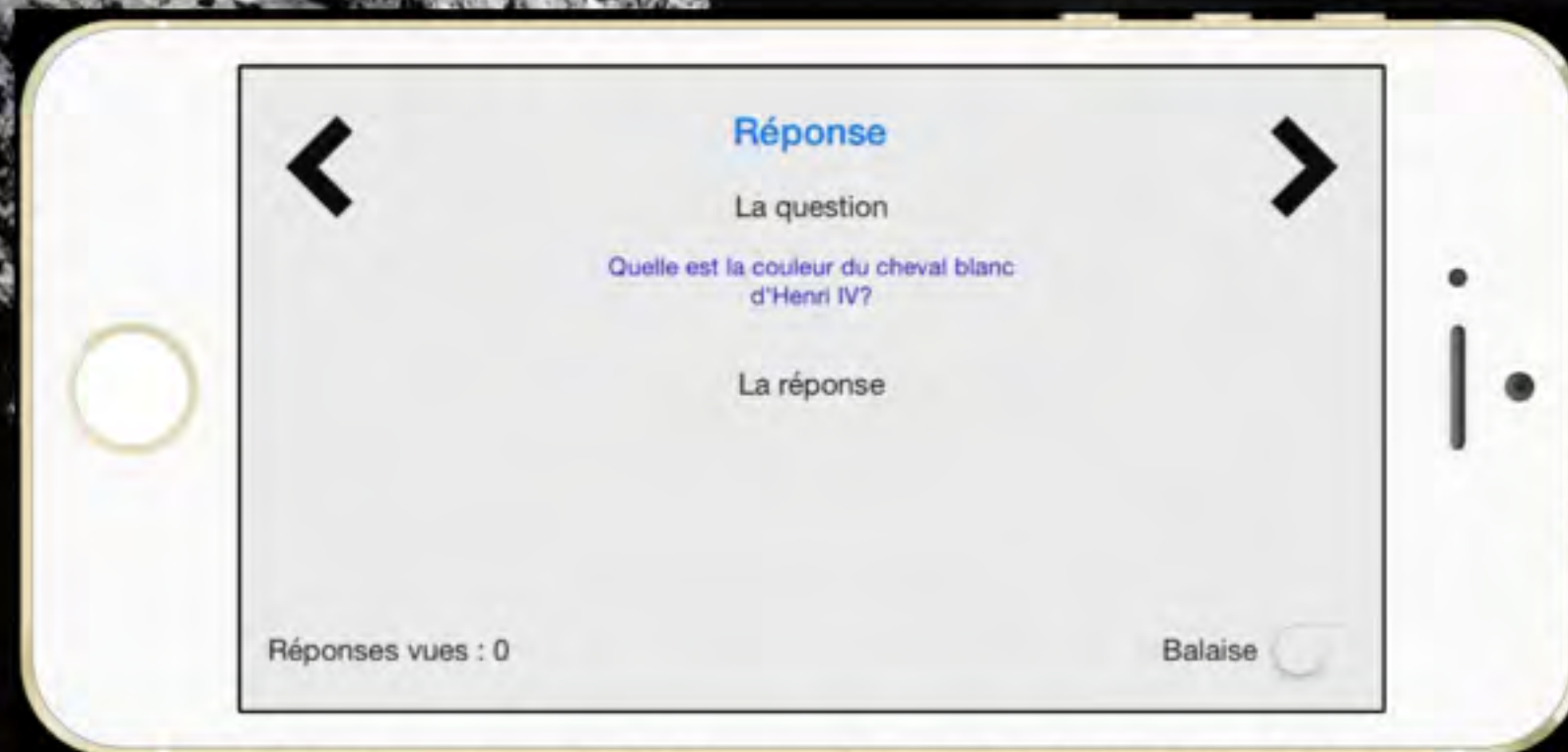
Et en mode horizontal?

101



Et en mode horizontal?

101



Quelques éléments

Les mécanismes que vous utiliserez

- UIButton + UILabel + UISwitch
- UITextView
 - ▶ En affichage seulement

Plus du code

- Classe QuestionsReponses
- Tableau de tuples
 - ▶ difficulté de la question
 - ▶ La question
 - ▶ La réponse
- Méthodes d'accès

Bref, de quoi...



En guise de conclusion...

103

Amusez-vous bien!!!



Vous pouvez enrichir l'exercice si cela vous chante!

En guise de conclusion...

Amusez-vous bien

34 + 80 LOC
«facile donc»



Vous pouvez enrichir l'exercice si cela vous chante!

A large, dark, irregularly shaped rock or meteorite is shown against a black background. The rock has a rough, textured surface with various shades of gray and black, suggesting a porous or crystalline structure. It is positioned diagonally, with the top-left corner pointing towards the upper left and the bottom-right corner pointing towards the lower right. The lighting highlights the jagged edges and the uneven surface of the rock.

The end