SCIENCES SORBONNE UNIVERSITÉ

Swift, creating your own classes

Fabrice.Kordon@lip6.fr

LIP6

As an introduction…

Structure of a class

- Interface + implementation…expressed on a single file

Renovated syntax & semantics («modern» 😂)

- Inspired from OO languages
- Inspired from functional languages

- Includes
  - **Properties (as Objective-C)**
  - **Methods**
  - **Inheritance**
  - **Initializers**
  - **Finalizers (deinit)**

The Student class


```
class Student       {

    var number = 0
    var name = ""

}

let s1 = Student()
s1.name = "Tartempion"
s1.number = 1000
print("Hello \(s1.name), your number is \(s1.number)")
```

```
Hello Tartempion, your number is 1000
```

The Student class


```
class Student       {

    var number = 0
    var name = ""

}

let s1 = Student()
s1.name = "Tartempion"
s1.number = 1000
print("Hello \(s1.name), your number is \(s1.number)")
```

Getters/Setters?

Is'n't it more a writing convention?

```
Hello Tartempion, your number is 1000
```

The Student class


```
class Student       {

    var number = 0
    var name = ""

}

let s1 = Student()
s1.name = "Tartempion"
s1.number = 1000
print("Hello \(s1.name), your number is \(s1.number)")
```

Getters/Setters?

Is'n't it more a writing convention?

If I want it?

notion of «computed properties»

```
Hello Tartempion, your number is 1000
```

Acces control to properties


```
class Student       {
    private var number = 0
    private var name = ""

    init (nm : String, nu : Int) { // Now required (no access to properties)
        name = nm
        number = nu
    }

    func myName() -> String { // Now required (no access to properties)
        return name
    }

    func myNumber() -> Int { // Now required (no access to properties)
        return number
    }
}

let s1 = Student(nm: "tartempion", nu:1000)
print("Hello \(s1.myName()), your number is \(s1.myNumber())")
```

```
Hello Tartempion, your number is 1000
```

Access control

Public or open

- Keywords open or public
  - **public : subclass and override within the defining module**
  - **open : subclass and override anywhere**
- Normal access (dotted notation)

Private

- Keyword private
- Only in the class body and its extensions

Only in the file

- Keyword fileprivate
- Only in the entities located in the same file

Internal (default mode)

- Keyword internal
- Only in the same module (e.g. App or framework)

Access control

Public or open

- Keywords open or public
  - **public : subclass and override wit**
  - **open : subclass and override an**
- Normal access (dotted nota

Private

- Keyword private
- Only in the class si s

Only in the fi

- Keyword filepriv
- Only in the entities loc he same file

Internal (default mode)

- Keyword internal
- Only in the same module (e.g. App or framework)

Properties & initializers

When attributes have a default value

- No need to write an init function

When declaration does not define a value

- You must have an initializer
- If not, the compiler complains

Computed attributes


```
class Student       {
    private var number = 0
    private var name = ""

    init (nm : String, nu : Int) { // Now required (no access to properties)
        name = nm
        number = nu
    }

    var identity : String { // Computed property
        get {
            return "\(name) (\(number))"
        }
    }

    func myIdentity () -> String { // Dedicated method
        return "\(name) (\(number))"
    }
}

let s1 = Student(nm: "tartempion", nu:1000)
print("Hello \(s1.identity)")
print("Hello \(s1.myIdentity())")
```

```
Hello tartempion (1000)
Hello tartempion (1000)
```

Lazy properties

Lazy evaluation

- Only when the value is needed
  - «var» properties only
  - Cannot be applied to computed properties

```
class DataImporter {
    // Import data from an external file (init takes time)
    var fileName = "data.txt"
    // functions to import data
}

class DataManager {
    lazy var importer = DataImporter()
    var data = [String]()
    // functions to manage data
}

let manager = DataManager()
manager.data.append("Some data")
manager.data.append("Some more data")
// «manager» exists but not its «importer» property
// to be elaborated when it is first accessed
```

Comparing references

- «=» everywhere

- =
  - Assignment
- ==, !=
  - Comparing values (e.g. memory areas)
  - «equal to»
- ===, !==
  - Comparing references
  - «has the same reference than»

Comparing references, some examples


```
class Student       {
    private var number = 0
    private var name = ""

    init (nm : String, nu : Int) { // Now required (no access to properties)
        name = nm
        number = nu
    }
}

let s1 = Student(nm : "Tartempion", nu :1000)
let s2 = Student(nm : "Tartempion", nu :1000)
let s3 = s1
if s1 === s2 {
    print ("This is the same reference")
} else {
    print ("There are two references")
}
if s1 === s3 {
    print ("This is the same reference")
} else {
    print ("There are two references")
}
```

```
There are two references
This is the same reference
```

As a conclusion

Just a small tour on classes

- Defining properties and methods
- handling visibility of properties and methods
- Computed properties
- Lazy properties
- References comparisons

More to learn…