第四周 代理人實作
UITableView

申潤五 Danny Shen
shenfive@gmail.com

今天的目標

物件導向 Inheritance，Overriding，Extension

Setter & Getter

協議與代理人 Protocol

UIPickerView

UITableView

Inheritance 繼承

師父會的，我秒會

師父的師父會的，我當然也很可能會，除非師父改良過，否則我只會師父那一個方法，不會師父的師父的方法

除了祖師爺，所有的人都有師父

如果不知道師父是誰，就以祖師爺當師父

Setter & Getter

物件的屬性，有時不會是一個變數，而是另一個變數計算而來，此時該屬性就可以設 Getter 與 Setter

```
class Year{
   var thisYear = 2022
   var 民國年:Int {
      set{
         thisYear = newValue + 1911
      }
      get{
         return thisYear - 1911
      }
   }
}
```

```
class Person{
   var _age = 0
   var age:Int  {
      set{
         _age = newValue
      }
      get{
         return _age
      }
   }
   func newYear(){
      _age += 1
   }
}
```

繼承(inherit)

這個概念主要是類別(class)在使用，是物件導向的一個重要特性。一個類別可以繼承另一個類別的屬性(property)、方法(method)及其他特性。一個類別繼承其他類別時，這個類別會被稱為子類別(subclass)。被繼承的類別則是稱為父類別(superclass，直翻會是超類別，但還是以父類別為主，以與子類別相對應)。

基礎類別

基礎類別(base class)就是不繼承於其它類別的類別。Swift 中沒有一個通用的基礎類別，只要一個類別沒有繼承於其他類別，這個類別即為一個基礎類別。

以下先定義一個基礎類別，其實就是一個普通的類別：

```
// 定義一個遊戲角色職業通用的類別

class GameCharacter {
    // 攻擊速度
    var attackSpeed = 1.5

    // 這個職業的敘述
    var description: String {
        return "職業敘述"
    }
    // 執行攻擊的動作
    func attack() {
        // 無任何動作 有待子類別實作
    }
}
```

```
// 生成一個類別 GameCharacter 的實體
let oneChar = GameCharacter()

// 印出：職業敘述
print(oneChar.description)
```

上述程式為一個基礎的遊戲角色職業類別，僅是定義了一些通用的內容，接著需要再定義子類別來完備。

生成子類別

生成子類別(subclassing)指的是基於一個基礎類別來定義一個新的類別，子類別會繼承父類別所有的特性，且還可以增加新的特性。

使用方式為在類別名稱後面加上冒號:，接著寫上父類別名稱，如下：

```
class 子類別: 父類別 {
   子類別的定義內容
}
```

以下是個例子，這邊定義一個繼承自類別GameCharacter的新類別Archer：

```
class Archer: GameCharacter {
   // 新增一個屬性 攻擊範圍
   var attackRange = 2.5
}
```

```
// 父類別所有的特性都一併繼承下來
let oneArcher = Archer()
// 可以直接以點語法來存取或設置一個父類別中定義過的屬性
oneArcher.attackSpeed = 1.8
```

一個子類別仍然可以再被其他類別繼承，以下再定義一個類別Hunter，繼承自類別Archer：

```
class Hunter: Archer {
   // 新增一個方法 必殺技攻擊
   func fatalBlow() {
      print("施放必殺技攻擊！")
   }
}

let oneHunter = Hunter()

// 這個類別一樣可以使用 Archer 及 GameCharacter 定義過的屬性及方法
print("攻擊速度為 \(oneHunter.attackSpeed)")
print("攻擊範圍為 \(oneHunter.attackRange)")

// 當然自己新增的方法也可以使用
oneHunter.fatalBlow()
```

Overriding

師父會的方法，我都會

但我也可以改良方法，或改變方法

師父叫的方法 super 方法

不管改良或改變，都叫 overrriding 那個方法

改良就是先做 super 的方法，再做自己的方法

改變就是直接做自己的方法

類別繼承的同時，子類別可以重新定義父類別中定義過的特性，如實體方法(instance method)、型別方法(type method)、實體屬性(instance property)、型別屬性(type property)或下標(subscript)，這種行為即是覆寫(overriding)。
使用關鍵字override來表示你要覆寫這個特性(即方法、屬性或下標)。
覆寫方法

以下為一個覆寫方法的例子：

```
// 使用並改寫前面定義的類別 Hunter
class OtherHunter: Archer {
   // 覆寫父類別的實體方法
   override func attack() {
      print("攻擊！這是獵人的攻擊！")
   }

   // 省略其他內容
}

let otherHunter = OtherHunter()
otherHunter.attack()
// 即會印出覆寫後的內容：攻擊！這是獵人的攻擊！
```

覆寫屬性

覆寫屬性時，需要使用getter(以及有時可省略的setter)來覆寫繼承來的屬性，且一定要寫上屬性的名稱及型別，這樣才能確定是從哪一個屬性繼承而來的。可以將一個繼承來的唯讀屬性覆寫為一個讀寫屬性，但不行將一個讀寫屬性覆寫為唯獨屬性。即原本有setter的話，覆寫時就一定要有setter。
以下是一個例子：

```
// 使用並改寫前面定義的類別 Hunter
class AnotherHunter: Archer {
   // 覆寫父類別的屬性 重新實作 getter 跟 setter
      override init() {
       super.init()
       attackSpeed = 3.4
   }
   override var attackSpeed: Double {
      get{
         return super.attackSpeed
      }
      set{
         print(newValue)
               super.attackSpeed = newValue
      }
   }
}
```

覆寫屬性觀察器

覆寫屬性時，通常可以加上屬性觀察器(property observer)，但要注意當繼承的屬性為常數儲存型屬性或唯讀計算型屬性時，不能加上屬性觀察器，因為這兩者的屬性無法再被設置，所以willSet跟didSet對它們沒有意義。
覆寫時不能同時有setter跟屬性觀察器(willSet跟didSet)，因為setter中即可做到屬性觀察器的功能要求。
雖然說是覆寫，但如果覆寫的父類別屬性也有屬性觀察器，其實子類別跟父類別兩者的屬性觀察器都會被執行，例子如下：

```
// 使用並改寫前面定義的類別 Archer
class OtherArcher: GameCharacter {
   // 覆寫一個屬性 重新實作 getter 跟 setter
   override var attackSpeed: Double {
      willSet {
         print("OtherArcher willSet")
      }
      didSet {
         print("OtherArcher didSet")
      }
   }
}
```

```
// 使用並改寫前面定義的類別 Hunter
class SomeHunter: OtherArcher {
   // 覆寫一個屬性 重新實作 getter 跟 setter
   override var attackSpeed: Double {
      willSet {
         print("SomeHunter willSet")
      }
      didSet {
         print("SomeHunter didSet")
      }
   }
   // 省略其他內容
}
let someHunter = SomeHunter()
// 設置新的值 會觸發 willSet 跟 didSet
someHunter.attackSpeed = 1.8
// 依序會印出：
// SomeHunter willSet
// OtherArcher willSet
// OtherArcher didSet
// SomeHunter didSet
```

上述程式中可以知道，willSet觸發時，會先執行子類別的再來才是父類別的，而didSet則是相反，先執行父類別的再來才是子類別的。

Extension

擴展(extension)是 Swift 一個重要的特性，它可以為已存在的列舉、結構、類別和協定添加新功能，而且不需要修改該型別原本定義的程式碼

使用extension關鍵字來定義一個擴展，格式如下：

```
extension 某個型別 {
   新增的程式內容
}
```

當你對一個已存在的型別新增一個擴展之後，擴展的新功能可以立即給該型別的所有實體使用，即使這個實體在定義擴展前就已經生成了也是可以。
另外，擴展也可以讓一個已有的型別遵循一個或多個協定，格式就如同結構及類別一樣：

```
extension 某個型別: 協定, 另一個協定, 又另一個協定 {
   新增的程式內容
}
```

後面章節會正式介紹協定。

計算屬性

擴展可以對內建的型別增加計算實體屬性與計算型別屬性。下面例子為內建的Double型別增加了 3 個計算實體屬性，用來表示常見的距離單位：

```
extension Double {
   var km: Double { return self * 1_000.0 }
   var m: Double { return self }
   var cm: Double { return self / 100.0 }
}
```

定義好新增的擴展之後，就可以直接使用，使用方法就如同普通的屬性一樣使用點語法再緊接著屬性名稱，如下：

```
// 直接對型別 Double 的值取得屬性

let aMarathon = 42.km + 195.m

// 印出：馬拉松的距離全長為 42195.0 公尺
print("馬拉松的距離全長為 \(aMarathon) 公尺")
```

protocol

協定(protocol)是 Swift 一個重要的特性，它會**定義**出為了**完成某項任務或功能**所需的方法、屬性，協定本身不會實作這些任務跟功能，而僅僅只是表達出該任務或功能的名稱。這些功能則都交由**遵循**協定的型別來實作，列舉、結構及類別都可以遵循協定，遵循協定表示這個型別必須實作出協定定義的方法、屬性或其他功能。

這有點像是法令規章，你想開車，就必需要駕照，至於你是考上的，國際駕照換的，還是拿雞腿換的，只要有駕照，就能通過檢查駕照這一關。擔任守衛，就規定必需要有武器，至於你是用 AK47 還是拿拖把當武器都可以，只要符合有武器的條件即可

之後我們會在 UITableView, UIPickerView 透過協定來處理資料來源或代理人等問題

使用protocol關鍵字來定義一個協定，格式如下：

```
protocol 協定名稱 {
    協定定義的內容
}
```

要讓自定義的型別遵循協定，寫法有點像繼承，一樣是把協定名稱寫在冒號:後面，而要遵循多個協定時，則是以逗號,分隔每個協定，格式如下：

```
struct 自定義的結構名稱: 協定, 另一個協定 {
    結構的內容
}
```

定義一個協定

底下是一個例子：

```
// 定義一個協定 包含一個唯讀的字串屬性
protocol FullyNamed {
    var fullName: String { get }
}
```

```
// 定義一個類別 遵循協定 FullyNamed
struct Person: FullyNamed {
    // 因為遵循協定 FullyNamed
    // fullName 這個屬性一定要定義才行 否則會報錯誤
    var fullName: String
}
let joe = Person(fullName: "Joe Black")
print("名字為 \(joe.fullName)")
// 印出：名字為 Joe Black
```

UIPickerView

是一種選擇器

使用方法有點類似我們之前使用的 Date Picker

必需透過協定來決定顯示的內容，也需要透過協定來決定選後要做的事情

一個 UIPickerView 通常會使用兩個協定

UIPickerViewDataSource 和
UIPickerViewDelegate

建立一個新的專案

在 ViewControler 中間拉一個 UIPickerView，設一下UI, 拉成 UIOutlet 取名 myPickerView，然後在 ViewDidLoad 中，加入

```
myPickerView.dataSource = self
```

這是說，希望我們用這個 ViewController 來訂義資料來源協定，但寫完會出錯，因為我們的 ViewControll 尚未繼承 UIPickerViewDataSource 協議
所以 class 要改成

```
class ViewController: UIViewController,UIPickerViewDataSource {
```

但是改好了之後，Xcode 又出現問題了，因為 UIPickerViewDataSource 必需實作兩個方法，所以我們要實作兩個方法，並給予暫時的值，Xcode 才不會出錯

```
  func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
  }

  func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int)
-> Int {
    return 1
  }
```

UIPickerViewDataSouce

設定 Picker View 的各項數字資料的協議，協議像是一問一答的過程，像是問該有幾個欄，每一個有幾個內容可選，每一個內容是什麼等

如 numberOfComponents(in pickerView: 回應有幾欄

pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: 則是回應欄有幾個選項

這通常配合集合資料來回應資料，讓我們實作一個輸入星座血型的 PickerView

UIPickerViewDelegate

這個協定處理 UIPickerView 的所有可見視圖與事件回應

所以看到的數量雖是 Data Souce 決定，看見什麼就是 Delegate 決定了

以 UIPickerView 來說，有數種不同的顯示方式，我們只需選一種就好了，例如單純的文字就可以用 pickearView(_ pickerView: UIPickerView, titleForRow

另外 UIPicerView 也不能像是 UIDatePicker 或 UIButton 般拉成 IBAction，事件發生也需 Delegate 來處理

首先，我們需要有 星座的陣列

```
var astrological = ["請選擇你的星座","白羊宮","金牛宮","雙子宮","巨蟹宮","獅子宮","處女宮","天秤宮","天蠍宮","射手宮","摩羯宮","水瓶宮","雙魚宮"]
var bloudType = ["請選擇你的血型","A","B","O","AB"]
```

所以，我們有兩種資料要輸入

```
//每一欄有幾個項目（欄）
func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 2
}
```

```
//每一欄要有幾個選項
func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    switch component {
    case 0: //星座
        return astrological.count
    case 1: //血型
        return bloudType.count
    default:
        return  0
    }
}
```

我們此時運行程式，就得到了一個都是問號的輸入器

如同 UIPickerView ，我們接上 Picker View Delegate，擴展 ViewController 最後加 pickerView(_ pickerView: UIPickerView, titleForRow 方法回應星座血型文字，這個階段的程式就是如下

```
import UIKit

class ViewController:
UIViewController,UIPickerViewDataSource,UIPickerViewDelegate {
======== 略 ==========
  func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
forComponent component: Int) -> String? {
      switch component {
      case 0: //星座
         return astrological[row]
      case 1: //血型
         return bloudType[row]
      default:
         return  nil
      }
  }
```

使用程式呼叫 Segue

除了由 UIView 直接拉之外，也可以由程式呼叫 Segue

1. Segue 是由 ViewController 拉到 ViewController
2. Segue 需要設定 ID
3. 使用 performSegue(withIdentifier: 就可以前往下一頁了
4. 前往前可於 prepare(for segue: 在進入下一個畫面做先處理例如傳送資料到下一頁

我們可以用這種方法做一個較完整的算命程式

設好 Segue 與 第二個 VievController 加入一個訊息 UILabel, 在原來的輸入星座血型頁面中，加入：

```
  // 使用者完成選擇時
  func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int) {
    if pickerView.selectedRow(inComponent: 0) != 0
      && pickerView.selectedRow(inComponent: 1) != 0{
      performSegue(withIdentifier: "gotoSecondPage", sender: nil)
    }
  }

  //進入下一個畫面前的處理
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "gotoSecondPage"{
      if let nextVC = segue.destination as? SecondViewController{
        nextVC.msg = "\(astrological[myPickerView.selectedRow(inComponent:
0)]) \n \(bloudType[myPickerView.selectedRow(inComponent: 1)]) 型 \n 一定發大
財"
      }
    }
  }
```

之後在第二頁顯示訊息就完成了

```
class SecondViewController: UIViewController {
  @IBOutlet weak var label: UILabel!
  var msg = ""

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    label.text = msg
  }
}
```

UITableView 簡介

UITableView 是商用 APP 中使用率最高的元件。當你需要將一批資料逐列顯示時常會使用到，每一個儲存格稱作一個 cell ，每個 cell 除了可以顯示文字外，還可以放置多個不同的元件

像是右側的畫面，實際上它是【一個】

UITableView 而已，那我們就來試試看吧

UITableView 開發項目

通常 UITableView 需要連結 UITableViewDelegate 與 UITableViewDataSource 兩個協定

UITableView 的顯示元件較複雜，不但有分 Section, Row, 還有 Header，Footer，等，但主要項目，還是 UITableViewCell，而我們通常要建立繼承這個 class 的新 class 用來設計自己的顯示方式

因為 UITableView 可以顯示很長的資料，甚至數百上千筆資料，如 Line 訊息或 Facebook 內容，所以 Cell 的產生，會利用之前的 Cell, 同一時間只產出畫面上顯示的資料

另外 UITableView 在實用時經常需要更新，不像 UIPicker View 只有一個畫面，所以操作起來非常複雜

常用的 DataSorce 與 Delegate

numberOfSections(in tableView: 回應 section 數量

tableView(_ tableView: UITableView, numberOfRowsInSection 回應每 sectionRow 數量

tableView(_ tableView: UITableView, cellForRowAt 回應顯示內容

tableView(_ tableView: UITableView, didSelectRowAt 回應點選事件

常用的 DataSorce 與 Delegate

tableView(_ tableView: UITableView, heightForRowAt 回答 cell 的高度

func tableView(_ tableView: UITableView, viewForHeaderInSection 回答 header 的顯示

tableView(_ tableView: UITableView, viewForFooterInSection 回答 footer 顯示

tableView(tableView: UITableView, editActionsForRowAtIndexPath 回答右滑編輯

實作範例 ~ 通訊錄

一個 Tab APP ，有兩個 Tab，第二個 Tab

先實作第二個 tab , 輸入分類，名稱和電話，並利用存到 APP 中

第一個 Tab，由讀取數據後

step 1 依分類排序整齊顯示

step 2 可以打電話或著刪除

學習如何傳資料，與不同 APP 之間的通訊協定