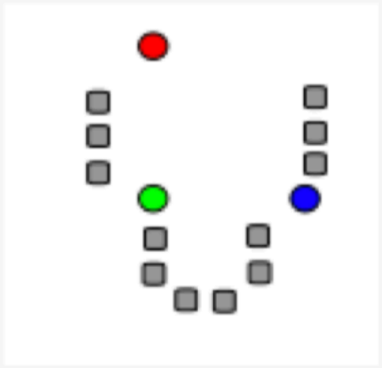
My KMeans Clustering II

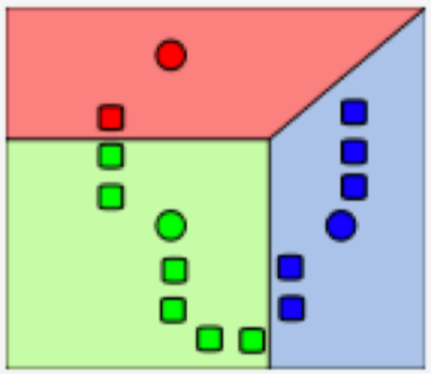
```
    @IBAction func strat3Clustering(_ sender: Any) {
        if KMeans.vectors.count < 3 {
            return
        }
        // setButtonEnable(false)
        KMeans.clusteringNumber = 3
        KMeans.clustering(500) { [unowned self] (success, centroids, clusters) -> () in
            if success {
                for point:UIView in self.spaceView.subviews {
                    point.removeFromSuperview()
                }
                for index in 0...2 {
                    for vector in clusters[index] {
                        let pointFrame = CGRect(x: CGFloat(vector[0]) - 5, y: CGFloat(vector[1]
- 5, width: 10.0, height: 10.0)
                        let point = UIView(frame: pointFrame)
                        if index == 0 {
                            point.backgroundColor = UIColor(red: 1, green: 0, blue: 0, alpha: 1
                        } else if index == 1 {
                            point.backgroundColor = UIColor(red: 0, green: 0, blue: 1, alpha: 1
                        } else {
                            point.backgroundColor = UIColor(red: 0, green: 1, blue: 0, alpha: 1
                        }
                        self.spaceView.addSubview(point)
                    }
                }
            }
            //self.setButtonEnable(true)
        }
    }
```

The algorithm is deemed to have converged when the assignments no longer change.

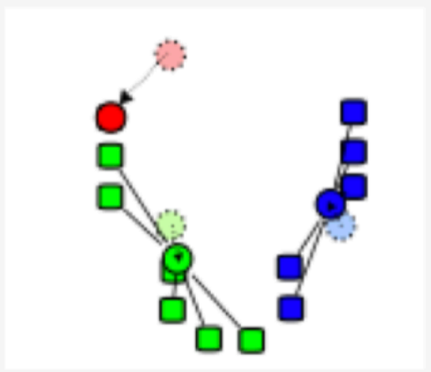
Demonstration of the standard algorithm



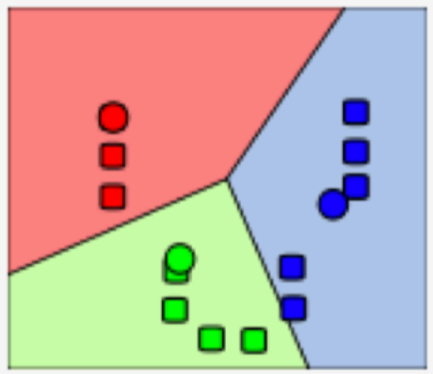
1) *k* initial "means" (in this case *k*=3) are randomly selected from the data set (shown in color).



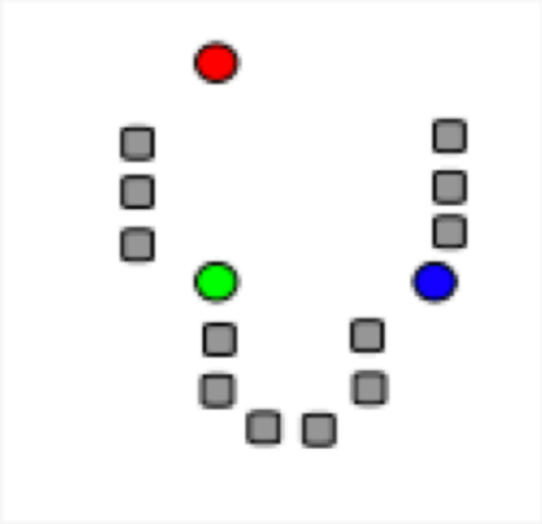
2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



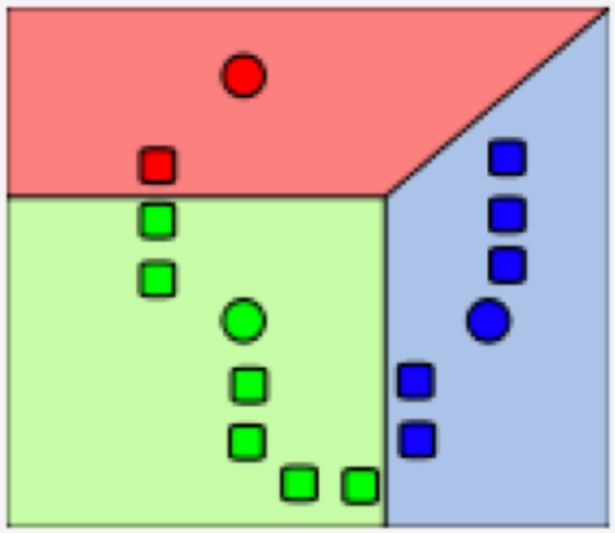
3) The centroid of each of the *k* clusters becomes the new means.



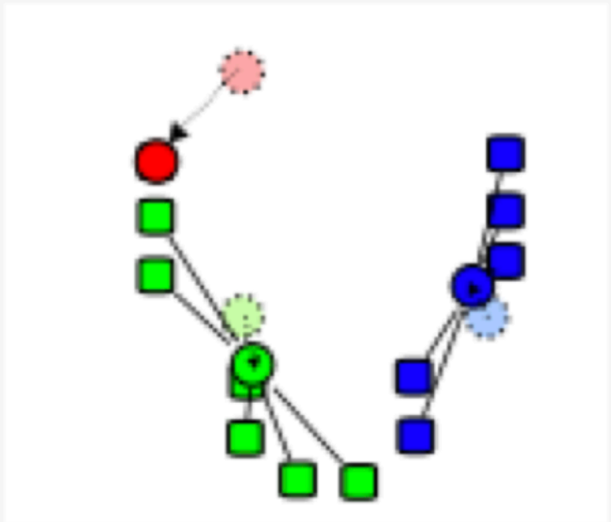
4) Steps 2 and 3 are repeated until convergence has been reached.



1) *k* initial "means" (in this case *k*=3) are randomly selected from the data set (shown in color).



2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3) The centroid of each of the *k* clusters becomes the new means.

4) Steps 2 and 3 are repeated until convergence has been reached.

```
typealias KMVectors = Array<[Double]>
typealias KMVector = [Double]

enum KMeansError: Error {
    case noDimension
    case noClusteringNumber
    case noVectors
    case clusteringNumberLargerThanVectorsNumber
    case otherReason(String)
}
```

```
class KMeansSwift {

    static let sharedInstance = KMeansSwift()
    fileprivate init() {}

    //MARK: Parameter

    //dimension of every vector
    var dimension:Int = 2
    //clustering number K
    var clusteringNumber:Int = 2
    //max interation
    var maxIteration = 100
    //convergence error
    var convergenceError = 0.01
    //number of excution
    var numberOfExcution = 1
    //vectors
    var vectors = KMVectors()
    //final centroids
    var finalCentroids = KMVectors()
    //final clusters
    var finalClusters = Array<KMVectors>()
    //temp centroids
    fileprivate var centroids = KMVectors()
    //temp clusters
    fileprivate var clusters = Array<KMVectors>()
```

Access Control

Access control restricts access to parts of your code from code in other source files and modules. This feature enables you to hide the implementation details of your code, and to specify a preferred interface through which that code can be accessed and used.

在swift中會有Access Control的概念形成，其實很好理解。想像一下，今天apple寫好了一個class，內部含有許多函數和屬性供你使用，但一定不希望使用者隨意修改程式碼造成系統大亂，因此只暴露出一些接口，即所謂的API （Application Programming Interface、應用程式介面），供外部使用這個class所提供的功能。

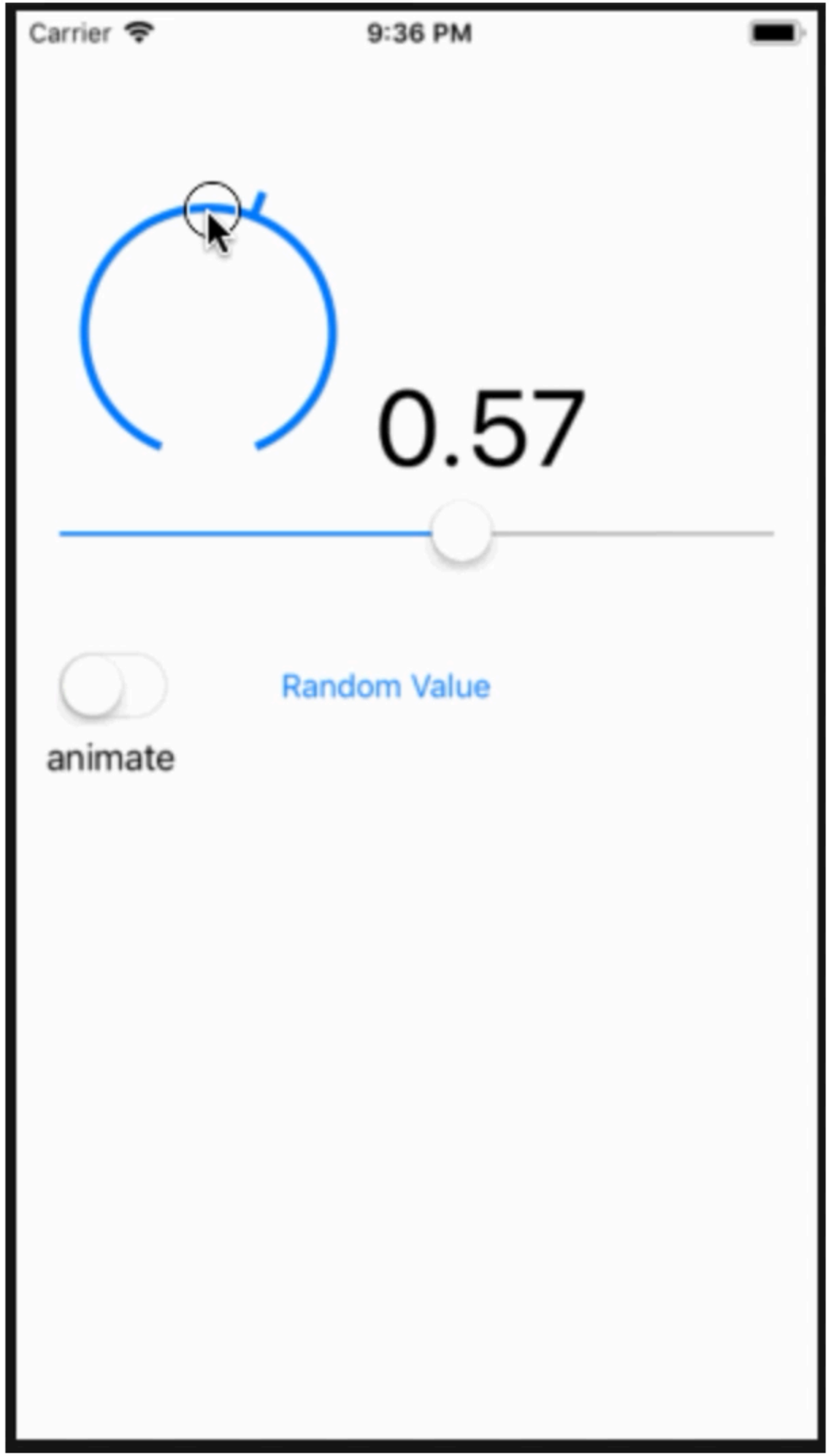
資料來源，網路

Framework for iOS

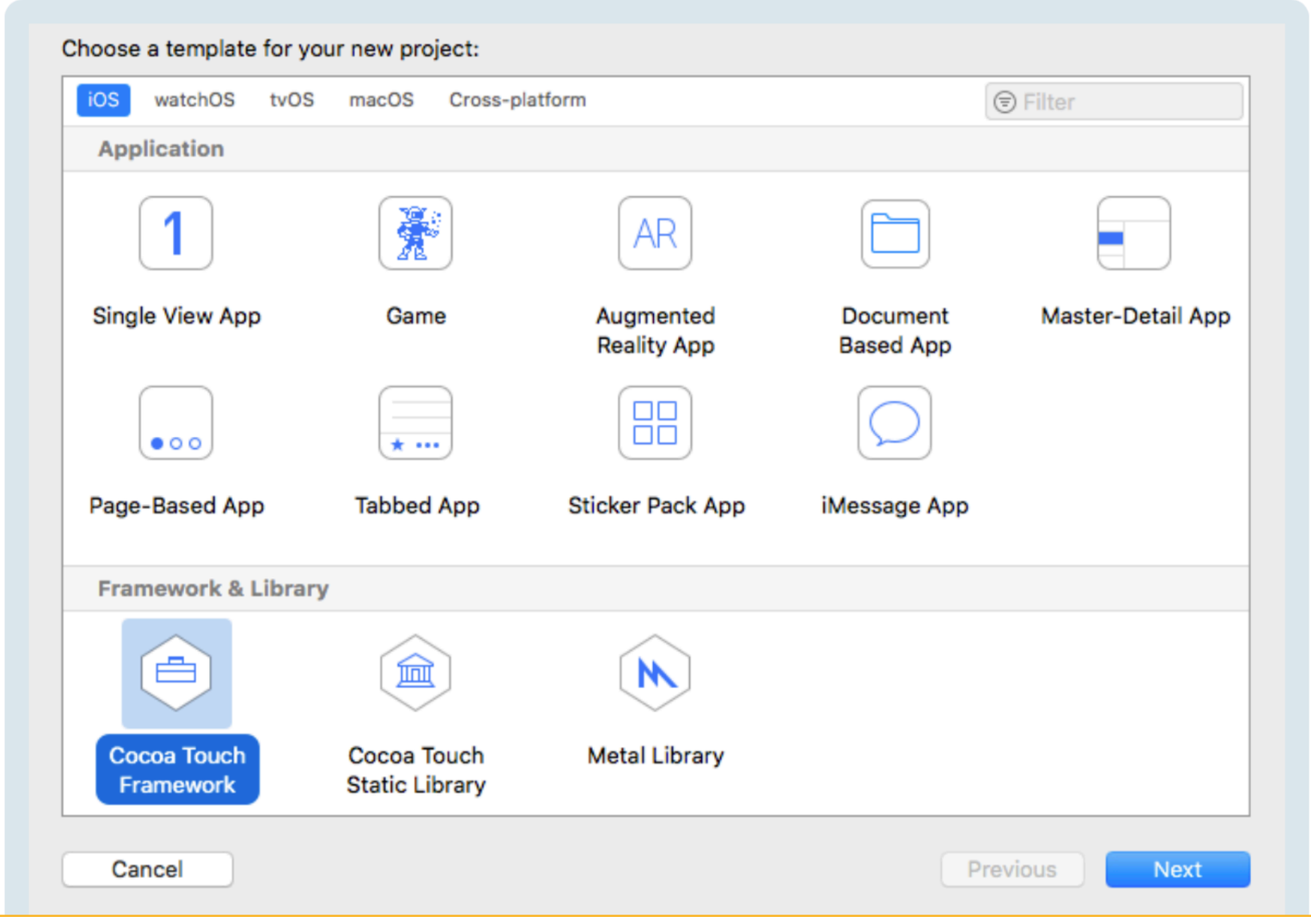
In this tutorial, you'll learn how to build an iOS framework so you can share code between apps, modularize your code, or distribute it as a third-party library.

By Lorenzo Boaro

Jul 11 2018 · Article (30 mins) · Beginner


Carrier
9:36 PM
0.57
Random Value
animate

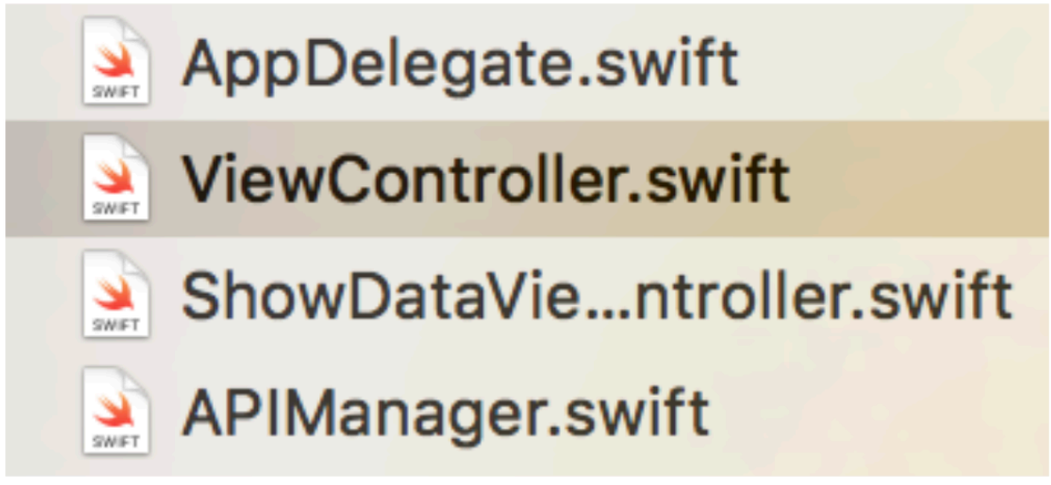
1. In Xcode, select **File ▸ New ▸ Project...**.
2. Choose **iOS ▸ Framework & Library ▸ Cocoa Touch Framework**.



> A module is a single unit of code distribution—a framework or application that is built and shipped as a single unit and that can be imported by another module with Swift's import keyword.
>
> Each build target (such as an app bundle or framework) in Xcode is treated as a separate module in Swift. If you group together aspects of your app's code as a stand-alone framework—perhaps to encapsulate and reuse that code across multiple applications—then everything you define within that framework will be part of a separate module when it's imported and used within an app, or when it's used within another framework.

Module ： 一個module可代表一個bundle ID下的app，一個framework。因此在一個app中，import了很多framework，每一個framework內即一個module，framework外的世界，也就是你自己撰寫的這個app，也是一個module。

Source Files ：就是每一個.swift檔案。舉例來說，下面圖內就有4個source file。



你撰寫的Access Control即影響了module和module間，module和source file間，source file和source file間的所有溝通情況。

資料來源，網路

Open *access* and *public access*

Open access and *public access* enable entities to be used within any source file from their defining module, and also in a source file from another module that imports the defining module. You typically use open or public access when specifying the public interface to a framework. The difference between open and public access is described below.

Internal access

同module內，不同swift檔案間access

Internal access enables entities to be used within any source file from their defining module, but not in any source file outside of that module. You typically use internal access when defining an app's or a framework's internal structure.

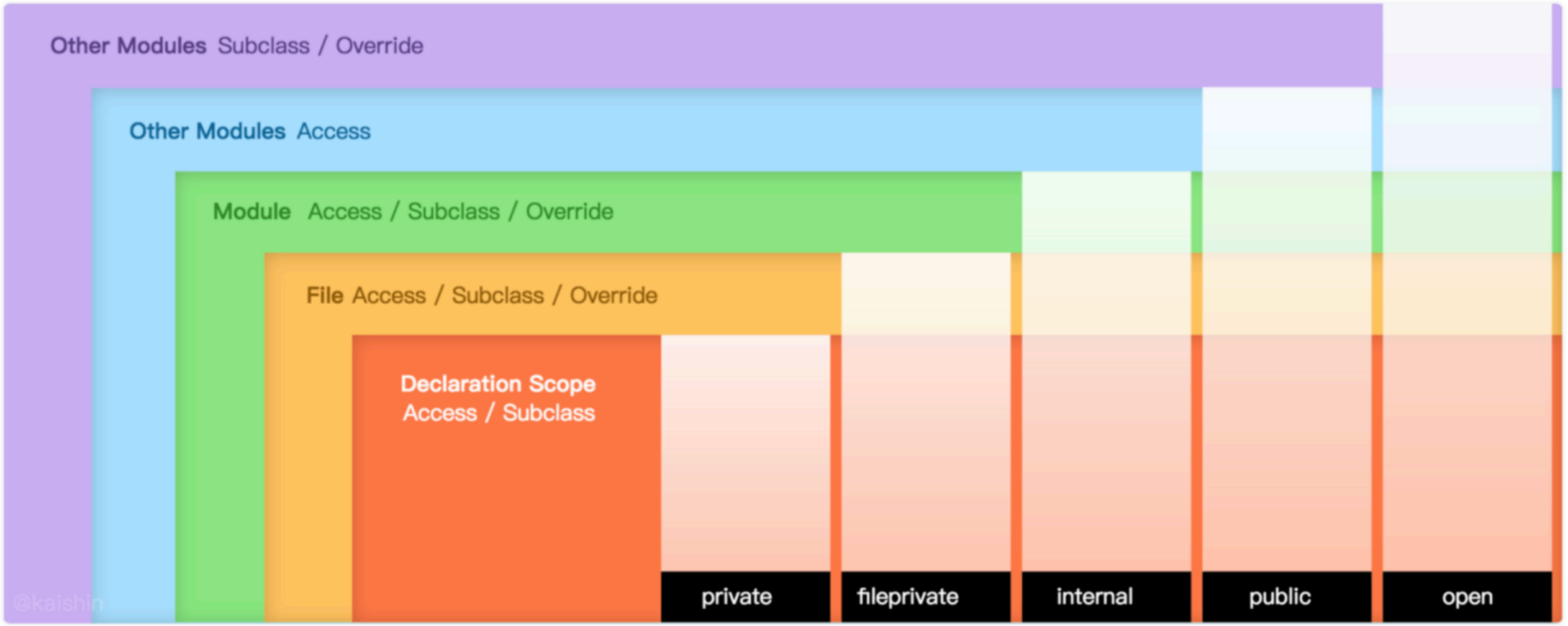
File-private access

File-private access restricts the use of an entity to its own defining source file. Use file-private access to hide the implementation details of a specific piece of functionality when those details are used within an entire file.

同swift檔案內access

Private access

Private access restricts the use of an entity to the enclosing declaration, and to extensions of that declaration that are in the same file. Use private access to hide the implementation details of a specific piece of functionality when those details are used only within a single declaration.


Other Modules Subclass / Override
Other Modules Access
Module Access / Subclass / Override
File Access / Subclass / Override
Declaration Scope
Access / Subclass
private
fileprivate
internal
public
open
@kaishin

資料來源，網路

Open access is the highest (least restrictive) access level and private access is the lowest (most restrictive) access level.

Open access applies only to classes and class members, and it differs from public access as follows:

- Open classes can be subclassed within the module where they're defined, and within any module that imports the module where they're defined.
- Open class members can be overridden by subclasses within the module where they're defined, and within any module that imports the module where they're defined.
- Classes with public access, or any more restrictive access level, can be subclassed only within the module where they're defined.
- Class members with public access, or any more restrictive access level, can be overridden by subclasses only within the module where they're defined.

在 Swift 中可分為 5 個層級。層級最高到最低依次為 open / public / internal / file-private / private。層級最高表示限制最少，反之表示限制最多。

1.open：

只用於 class 宣告以及 class 內所有的成員宣告。所有 module 和 source file 都可取得，可讓繼承者繼承此 class 和 override 此 class 內的 method。

2.public：

不限用於 class，所有 module 和 source file 都可取得。但若為 class，則繼承者不可繼承此 class 和override 其 method。

3.internal：

宣告於 module 內所有 source file 才可取得，於 module 外無法取得。

module 內所有source file，若為class，可讓繼承者繼承此 class 和 override 此 class 內的 method。同時，此為預設的 access control，也就代表預設的存取權限只能用於你寫的 module 內，在 module 外無法取得。

4.fileprivate：

宣告的 source file 內才可取得，若為 class，可讓繼承者繼承此 class 和 override 此 class 內的 method。

5.private ：

宣告的區塊內才可取得，也就是大括號內才可取得。若為 class，可讓繼承者繼承此 class 和 override 此 class內的 method。

資料來源，網路

EDIT

Let us have following class in module (project) *Module1*:

```
class A {
}
```

Since this class is not `open`, it can be subclassed only within the same module. That means that following class:

```
class B: A {
}
```

Can be written **only** in the same project, in *Module1*.

If you add *Module1* as a dependency to project *Module2*, and try to do this:

```
import Module1

class C: A {
}
```

It will not compile. That's because class `A` is not `open` (in other words it has access `public` or less) and it does not belong to the same module as `C`. `A` belongs to *Module1*, `C` belongs to *Module2*.

Note

`import` keyword imports a dependency module to your current module. If you write `import UIKit` in your project, you are telling the compiler that you want to use module `UIKit` in your module. `import` does not define current module. Current module is the current project.

Adding `import UIKit` at the beginning of the file does not change nor define to which module the file belongs. It just tells the compiler that in that file you want to use code from `UIKit` module.

Access Control Syntax

Define the access level for an entity by placing one of the `open`, `public`, `internal`, `fileprivate`, or `private` modifiers before the entity's introducer:

```
public class SomePublicClass {}
internal class SomeInternalClass {}
fileprivate class SomeFilePrivateClass {}
private class SomePrivateClass {}

public var somePublicVariable = 0
internal let someInternalConstant = 0
fileprivate func someFilePrivateFunction() {}
private func somePrivateFunction() {}
```

```
class SomeInternalClass {}                  // implicitly internal
let someInternalConstant = 0                // implicitly internal
```

```swift
public class SomePublicClass {                  // explicitly public class
    public var somePublicProperty = 0            // explicitly public class member
    var someInternalProperty = 0                 // implicitly internal class member
    fileprivate func someFilePrivateMethod() {}  // explicitly file-private class member
    private func somePrivateMethod() {}          // explicitly private class member
}

```

```swift
class SomeInternalClass {                          // implicitly internal
  class
    var someInternalProperty = 0                    // implicitly internal
  class member
    fileprivate func someFilePrivateMethod() {}  // explicitly file-private
  class member
    private func somePrivateMethod() {}             // explicitly private
  class member
}

fileprivate class SomeFilePrivateClass {        // explicitly file-private
  class
    func someFilePrivateMethod() {}              // implicitly file-private
  class member
    private func somePrivateMethod() {}          // explicitly private
  class member
}

```

```
19  private class SomePrivateClass {                    // explicitly private class
20      func somePrivateMethod() {}                      // implicitly private
      class member
21  }
```

```
class KMeansSwift {

    static let sharedInstance = KMeansSwift()
    fileprivate init() {}

    //MARK: Parameter

    //dimension of every vector
    var dimension:Int = 2
    //clustering number K
    var clusteringNumber:Int = 2
    //max interation
    var maxIteration = 100
    //convergence error
    var convergenceError = 0.01
    //number of excution
    var numberOfExcution = 1
    //vectors
    var vectors = KMVectors()
    //final centroids
    var finalCentroids = KMVectors()
    //final clusters
    var finalClusters = Array<KMVectors>()
    //temp centroids
    fileprivate var centroids = KMVectors()
    //temp clusters
    fileprivate var clusters = Array<KMVectors>()
```

```
//MARK: Public

 //check parameters
 func checkAllParameters() -> Bool {
        }

 //add vectors
 func addVector(_ newVector:KMVector) {
        }

 func addVectors(_ newVectors:KMVectors) {
        }
```

```
    //clustering
    func clustering(_ numberOfExcutions:Int, completion:(_ success:Bool, _
centroids:KMVectors, _ clusters:[KMVectors])->()) {
            }

    func reset() {
   }
```

```
    //MARK: Private

     // 1: pick initial clustering centroids randomly
     fileprivate func pickingInitialCentroidsRandomly() {
     }

     // 2: assign each vector to the group that has the closest centroid.
     fileprivate func assignVectorsToTheGroup() {
    }

     // 3: recalculate the positions of the K centroids. (return move distance
square)
     fileprivate func recalculateCentroids() -> Double {
    }

     // 4: repeat 2,3 until the new centroids cannot move larger than
convergenceError or the iteration is over than maxIteration
     fileprivate func beginClustering() -> Double {
     }

     // the cost function
     fileprivate func costFunction() -> Double {
      }

     // 5: excute again (up to the number of excution), then choose the best result
     private func beginClusteringWithNumberOfExcution(_ number:Int) {

}
```

```
//MARK: Helper

//Add Vector
private func vectorAddition(_ vector:KMVector, anotherVector:KMVector) -> KMVector {
}

//Calculate Euclidean Distance
private func EuclideanDistance(_ v1:[Double],v2:[Double]) -> Double {
}

private func EuclideanDistanceSquare(_ v1:[Double],v2:[Double]) -> Double {

}

//Extension to pick random number. According to stackoverflow.com/questions/27259332/
get-random-elements-from-array-in-swift
private extension Int {
}

private extension Array {
}
```

```swift
//check parameters
func checkAllParameters() -> Bool {
    if dimension < 1 { return false }
    if clusteringNumber < 1 { return false }
    if maxIteration < 1 { return false }
    if numberOfExcution < 1 { return false }
    if vectors.count < clusteringNumber { return false }
    return true
}
```

```
//add vectors
func addVector(_ newVector:KMVector) {
    vectors.append(newVector)
}

func addVectors(_ newVectors:KMVectors) {
    for newVector:KMVector in newVectors {
        addVector(newVector)
    }
}
```

```
    //clustering
    func clustering(_ numberOfExcutions:Int, completion:(_ success:Bool, _
centroids:KMVectors, _ clusters:[KMVectors])->()) {
        beginClusteringWithNumberOfExcution(numberOfExcutions)
        return completion(true, finalCentroids, finalClusters)
    }

    func reset() {
        vectors.removeAll()
        centroids.removeAll()
        clusters.removeAll()
        finalCentroids.removeAll()
        finalClusters.removeAll()
    }
```

```swift
// 1: pick initial clustering centroids randomly
fileprivate func pickingInitialCentroidsRandomly() {
    let indexes = vectors.count.indexRandom[0..<clusteringNumber]
    var initialCenters = KMVectors()
    for index:Int in indexes {
        initialCenters.append(vectors[index])
    }
    centroids = initialCenters
}
```

```
    fileprivate func assignVectorsToTheGroup() {
        clusters.removeAll()
        for _ in 0..<clusteringNumber {
            clusters.append([])
        }
        for vector in vectors{
            var tempDistanceSquare = -1.0
            var groupNumber = 0
            for index in 0..<clusteringNumber {
                if tempDistanceSquare == -1.0 {
                    tempDistanceSquare = EuclideanDistanceSquare(vector, v2:
centroids[index])
                    groupNumber = index
                    continue
                }
                if EuclideanDistanceSquare(vector, v2: centroids[index]) <
tempDistanceSquare {
                    groupNumber = index
                }
            }
            clusters[groupNumber].append(vector)
        }
    }
```

```
    // 3: recalculate the positions of the K centroids. (return move distance square)
    fileprivate func recalculateCentroids() -> Double {
        var moveDistanceSquare = 0.0
        for index in 0..<clusteringNumber {
            var newCentroid = KMVector(repeating: 0.0, count: dimension)
            let vectorSum = clusters[index].reduce(newCentroid, { vectorAddition($0,
anotherVector: $1) })
            var s = Double(clusters[index].count)
            vDSP_vsdivD(vectorSum, 1, &s, &newCentroid, 1, vDSP_Length(vectorSum.count))
            if moveDistanceSquare < EuclideanDistanceSquare(centroids[index], v2:
newCentroid) {
                moveDistanceSquare = EuclideanDistanceSquare(centroids[index], v2:
newCentroid)
            }
            centroids[index] = newCentroid
        }
        print("=====new centers=====")
        print(centroids)
        return moveDistanceSquare
    }
```

```
    // 4: repeat 2,3 until the new centroids cannot move larger than
convergenceError or the iteration is over than maxIteration
    fileprivate func beginClustering() -> Double {
        pickingInitialCentroidsRandomly()
        var iteration = 0
        var moveDistance = 1.0
        while iteration < maxIteration && moveDistance > convergenceError {
            iteration += 1
            assignVectorsToTheGroup()
            moveDistance = recalculateCentroids()
        }
        return costFunction()
    }
```

```
// the cost function
fileprivate func costFunction() -> Double {
    var cost = 0.0
    for index in 0..<clusteringNumber {
        for vector in clusters[index] {
            cost += EuclideanDistanceSquare(vector, v2: centroids[index])
        }
    }
    return cost
}
```

```
// 5: excute again (up to the number of excution), then choose the best result
private func beginClusteringWithNumberOfExcution(_ number:Int) {
    var number = number
    if number < 1 { return }
    var cost = -1.0
    while number > 0 {
        let newCost = beginClustering()
        if cost == -1.0 || cost > newCost {
            cost = newCost
            finalCentroids = centroids
            finalClusters = clusters
        }
        number -= 1
    }
}
```

```swift
//MARK: Helper

//Add Vector
private func vectorAddition(_ vector:KMVector, anotherVector:KMVector) -> KMVector {
    var addresult = KMVector(repeating: 0.0, count: vector.count)
    vDSP_vaddD(vector, 1, anotherVector, 1, &addresult, 1, vDSP_Length(vector.count))
    return addresult
}
```

```swift
//Calculate Euclidean Distance
private func EuclideanDistance(_ v1:[Double],v2:[Double]) -> Double {
    let distance = EuclideanDistanceSquare(v1,v2: v2)
    return sqrt(distance)
}
```

```swift
private func EuclideanDistanceSquare(_ v1:[Double],v2:[Double]) -> Double {
    var subVec = [Double](repeating: 0.0, count: v1.count)
    vDSP_vsubD(v1, 1, v2, 1, &subVec, 1, vDSP_Length(v1.count))
    var distance = 0.0
    vDSP_dotprD(subVec, 1, subVec, 1, &distance, vDSP_Length(subVec.count))
    return abs(distance)
}
```

```swift
//Extension to pick random number. According to stackoverflow.com/
questions/27259332/get-random-elements-from-array-in-swift
private extension Int {
    var random: Int {
        return Int(arc4random_uniform(UInt32(abs(self))))
    }
    var indexRandom: [Int] {
        return  Array(0..<self).shuffle
        }
}
```

```
private extension Array {
    var shuffle:[Element] {
        var elements = self
        for index in 0..<elements.count {
            let anotherIndex =
Int(arc4random_uniform(UInt32(elements.count - index))) + index
            anotherIndex != index ? elements.swapAt(index,
anotherIndex) : ()
        }
        return elements
}
mutating func shuffled() {
    self = shuffle
}
```