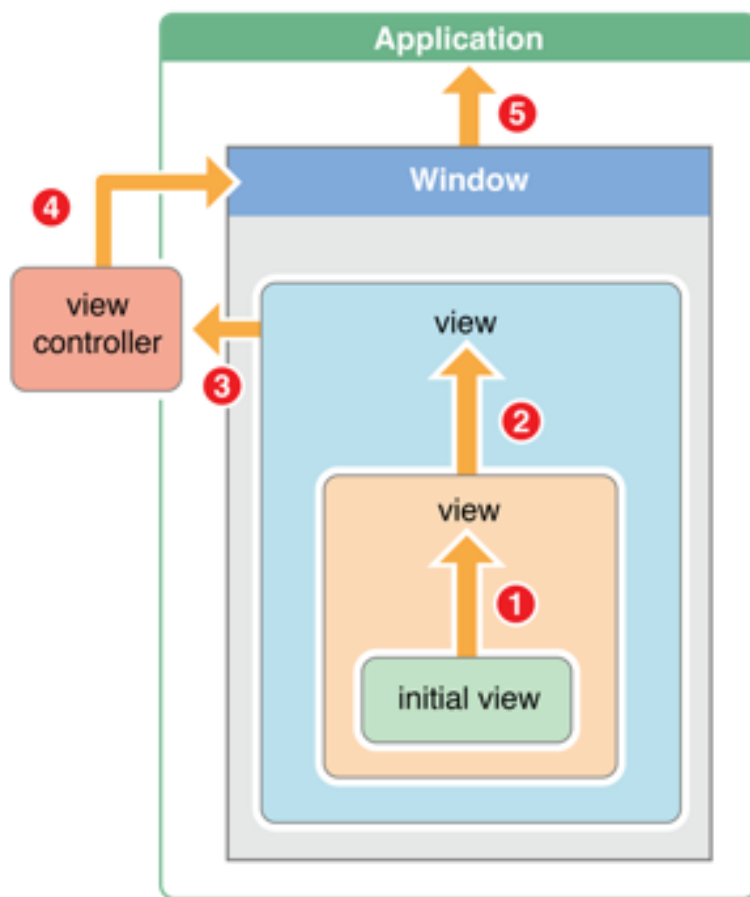


3rd Day - iOS Swift Training in Barcelona

The Responder Chain and the Focus Engine

In iOS there is a so called Responder Chain. When you touch an object view on the screen (the initial object) it can become the first responder if enabled to do it and then handles the touch event, otherwise UIKit passes the request to its VC or superview, and so on, up to the main Window and then to the UIApplication.



The first responder is the first object in a responder chain which is able to handle the event. In most cases, the first responder is a view object that the user selects or activates with the mouse or keyboard.

A standard UIView is not first responder by default. Buttons, Sliders, TextField, TextViews they are.

A similar chain exists in tvOS, and it is called Focus Engine. In this case the request propagates in the reverse direction, from the back UIWindow forward to the upfront object. The subviews are scanned from the top left most proceeding line by line. This is one of the most important differences between

iOS and tvOS. When the first responder object is reached it has a pulsing animation that focuses the user attention to it.

Text Field

Displays a rounded rectangle that can contain editable text. When a user taps a text field, a keyboard appears; when a user taps the Return key, the keyboard disappears and the text field can handle the input in an application-specific way.

UITextField supports overlay views to display additional information, such as a bookmarks icon. UITextField also provides a clear text control a user taps to erase the contents of the text field.

A Text Field has 3 small views inside. The left and right ones are optional. You can use them to add some symbols (for example the lens for a search field)

UITextFieldDelegate allows to add more functionality to the class.

Text View

Text Views are very similar, but they can scroll the text content up and down. You will use a Text View if the user can input a lot of text. UITextView can contain multiple lines of editable text.

For example if you need to edit an e-mail, use Text View.

If you touch and hold the text within a Text View you will trigger a text selection callback.

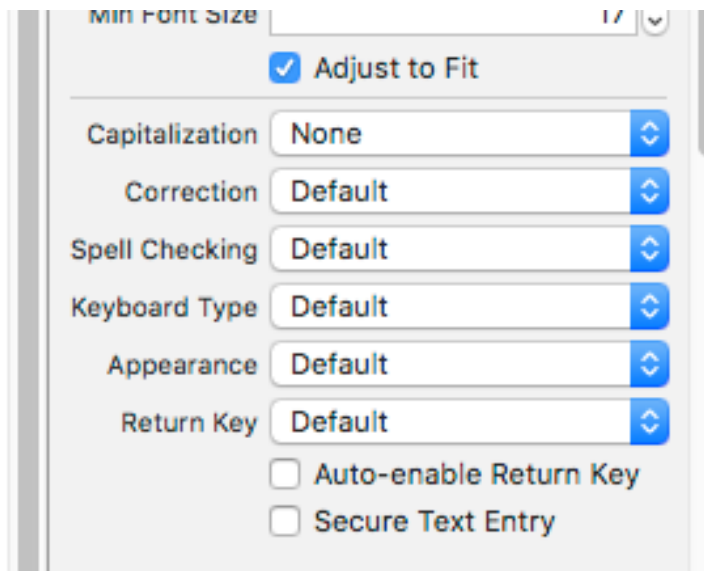
You can specify attributes, such as font, color, and alignment, that apply to all text in a Text View.

Keyboard

As for many other object views, when you touch a Text Field or a TextV View it reacts as a first responder by showing up the keyboard (the iPad is a little bit more flexible, because you can decide to show up something else, like an emoji).

By using the resignFirstResponder() delegate method you can remove the keyboard.

In IB you have a lot of settings for controlling the keyboard behaviour.



Attributed String

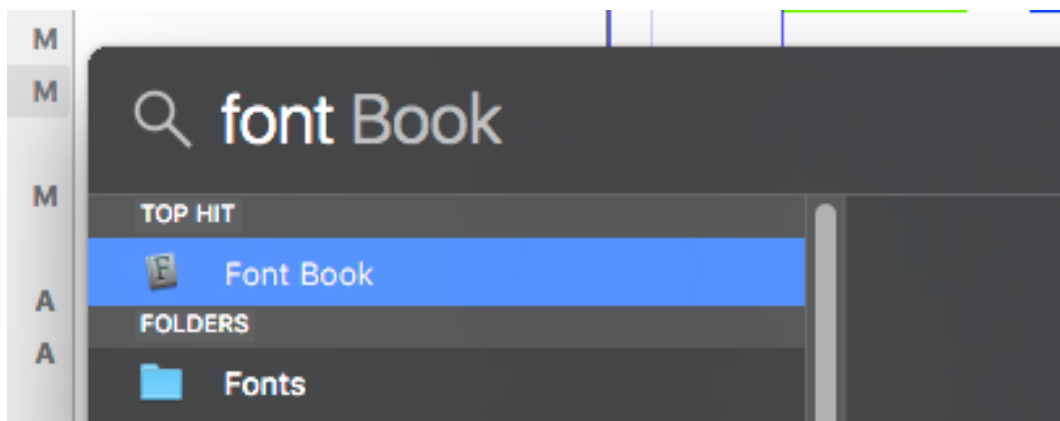
In the past, in order to customise text finely (size, fonts, colors, ..) you had to do a lot of work using html and a framework called CoreText.

Since iOS 7 you can use Attributed strings. Text can be Plain or Attributed. For example a UILabel label has a property called label.text and a property called label.attributedText.

NSAttributedString represents a string with attributes. The attributes are organised as a dictionary.

In iOS there are about 30 fonts, but you can add more, some for free, some under payment.

Tip: to use a mac font in iOS open Font Book and drag the font in the project (from Finder)



In order to edit the text attributes, always do it through the attribute inspector of the IB.



Attributed String Creator is an app that translates typed text into attributes.



For more complex text design you should use Core Text.

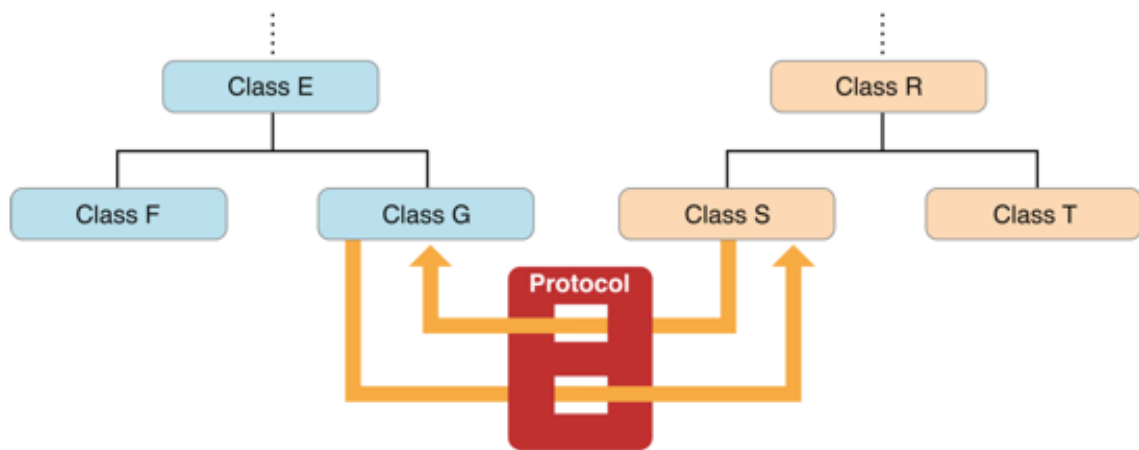
Delegation and Protocols

Delegation is a simple and powerful pattern in which one object (the delegate) in a program acts on behalf of, or in coordination with, another object (the delegating).

The delegating object is typically a framework object, and the delegate is typically a custom controller object.

The delegating object keeps a reference to the other object and at the appropriate time sends a message to it. The message informs the delegate of an event that the delegating object is about to handle or has just handled. The delegate may respond to the message by updating the appearance or state of itself or other objects in the app, and in some cases it can return a value that affects how an impending event is handled. In a managed memory environment, the delegating object maintains a weak reference to its delegate.

A protocol declares a programmatic interface that any class may choose to implement. Protocols make it possible for two classes distantly related by inheritance to communicate with each other to accomplish a certain goal. They thus offer an alternative to subclassing.



DELEGATION NEEDS PROTOCOLS, BUT PROTOCOLS ARE ALSO USED FOR OTHER PURPOSES.

How to pass data from a VC to another VC ?

IF YOU WANT TO SEND DATA FROM THE CHILD VC BACK TO THE PARENT VC YOU MUST USE THE DELEGATION DESIGN PATTERN.

IN THE FORWARD DIRECTION IT IS SIMPLE: JUST INITIALIZE SOME PROPERTIES BEFORE ENABLING THE VC.

In the following two examples we will use delegation and protocols.

Example: Login app

In this app we press a login button on the main view to present modally another view controlled by a login VC (nib).

```

11 class ViewController: UIViewController, LoginViewControllerDelegate {
12
13     @IBAction func login () {
14         let vc = LoginViewController()
15         vc.delegate = self // set the delegate, i.e. the main VC
16
17         // we present the login VC modally
18         present(vc, animated: true, completion: nil)
19     }
20
21     func login(viewController vc: LoginViewController, didFinishWith result: [String : String])
22     {
23         //we must always dismiss a VC at the same level where we presented it
24         dismiss(animated: true, completion: nil)
25
26         if let loginValue = result["Login"] {
27             print(loginValue)
28         }
29     }
30 }

```

The LoginViewController controls a TextField and defines a protocol which is used to notify to the main VC that the user has pressed the Return key on the keyboard.

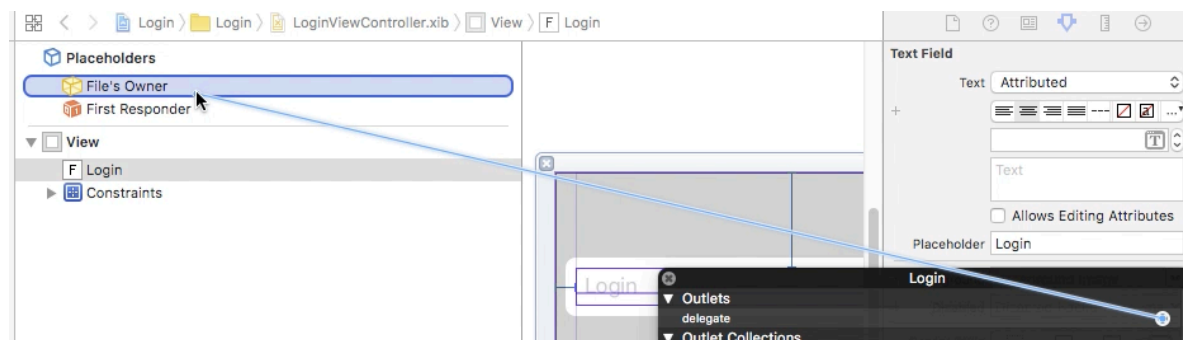
```

12 // we define a class only protocol (because we have set a weak reference, unknown to
    structures)
13 protocol LoginViewControllerDelegate : class {
14     // now let's write the list of empty methods of the protocol. Here just one
15     func login(viewController vc: LoginViewController, didFinishWith result: [String : String])
16 }
17
18 // comply with UITextFieldDelegate to remove the keyboard when user taps Enter
19 class LoginViewController: UIViewController, UITextFieldDelegate {
20
21     weak var delegate: LoginViewControllerDelegate? //we need it to get the reference to the
        delegate object (in our case it is the VC). weak is important in order to make the
        reference not strong otherwise we create a retain cycle
22
23     // protocol callback when the user tap the Enter key
24     func textFieldShouldReturn(_ textField: UITextField) -> Bool {
25         textField.resignFirstResponder() // removing the keyboard
26         // call the protocol function login, implemented by the main VC. We also pass the text
27         if let text = textField.text {
28             delegate?.login(viewController: self, didFinishWith: ["Login" : text])
29             //note the use of the optional chaining.
30         }
31         return true
32     }
33     // there are a number of such callback
34     func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
35         print("did begin editing")
36         return true
37     }
38 }

```

Here we use the Modal Presentation. We do it when we want to force the user to do something (for example login with password, etc.).

In order to make the keyboard to disappear by tapping the return key we need to use `resignFirstResponder()` within `textFieldShouldReturn`, a delegate method defined in the `UITextFieldDelegate`. To set the `LoginViewController` as the delegate of the `Login` `TextField` in the `xib`, go to the `xib` file, `Ctrl + click` on to the `TextView`, and then



When we dismiss the keyboard we also want to go back to the main VC using a custom protocol. This is because we can dismiss the login view only at the same level it was presented, in the VC. Dismissing the frontmost view controller is done through the call

```
dismiss(animated: true, completion: nil)
```

In the `Login` class we reference the delegate object (the main VC) as weak, not strong. This is because when we pressed the button the main VC created a strong reference to the login VC. We have to create a weak back reference

otherwise we have a retain cycle!

Creating a class only custom protocol is simple:

```
@objc protocol LoginViewControllerDelegate : class {  
  
    func a (..) {..}                // mandatory method  
    ..  
    @objc optional func b (..) {..} // optional method  
    ..  
}
```

The protocol can be created in a separated file. In Swift you can put everything in a file with any name. In the example above we decided to implement the protocol in the same file of the class definition.

Optional chaining

When we see something like

`objname?.method(..)`

the method is executed only if `objname` is not nil.

If also the method is optional we have

`objname?.method?(..)`

which will be executed if `objname` is not nil AND the method has been implemented by the delegate.

Example: Container app

In the `MapViewController` implement the class protocol, and define a class property as the weak reference to the delegate object

```
12 protocol MapViewControllerDelegate : class {  
13     // now let's write the list of empty methods of the protocol. Here just one  
14     func showDetailView(viewController vc: MapViewController, didFinishWith result: [String : String])  
15 }  
16  
17 class MapViewController: UIViewController, MKMapViewDelegate {  
18     weak var delegate: MapViewControllerDelegate? //we need it to get the reference to the delegate object (in our case it is the VC)  
19 }
```

Note the “: class” in the protocol definition. It states that the protocol is compatible only with classes, because we are going to use a weak reference to the delegate object, and “weak” is a concept that applies only to classes, not to structures. In other words this protocol is a class only protocol.

In the same class, in a suitable place, call the delegate method defined by the protocol and implemented by the delegate (you can also pass some data within

a dictionary)

```
82 delegate?.showDetailView(viewController: self, didFinishWith: ["someKey" : "someval"])
```

Note the use of the optional chaining **obj-name?.method-name** :the method is executed only if obj-name is not nil.

In the delegate class, declare conformity to the protocol

```
11 class ViewController: UIViewController, MapViewControllerDelegate {
```

and also prepare the strong reference to self (the delegate) needed by the delegating object

```
13 override func viewDidLoad() {  
14     super.viewDidLoad()  
15     let mapVC = MapViewController()  
16     mapVC.delegate = self // needed by the delegating object as a strong reference to the delegate object
```

and then implement the delegate method. In this case the method creates another VC and shows its view

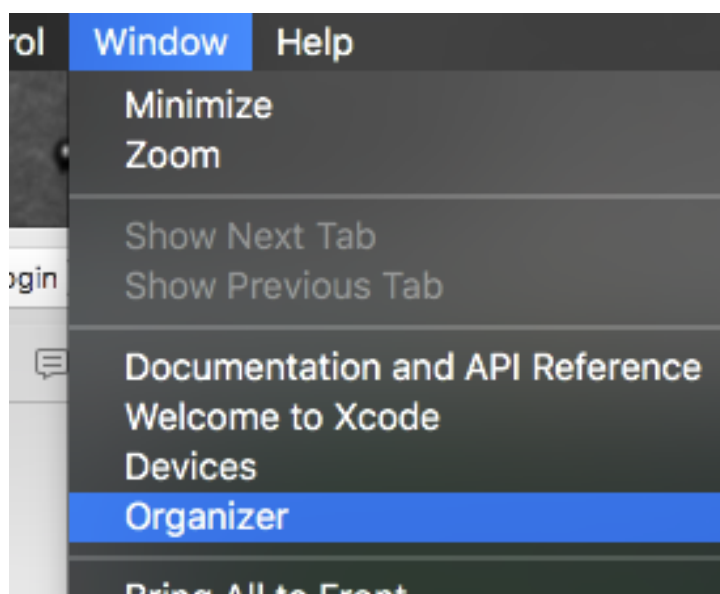
```
111 // implementation of the delegate method requested by the custom protocol MapViewControllerDelegate  
112 func showDetailView(viewController vc: MapViewController, didFinishWith result: [String : String]) {  
113     let detailVC = DetailViewController()  
114     show(detailVC, sender: nil)  
115 }
```

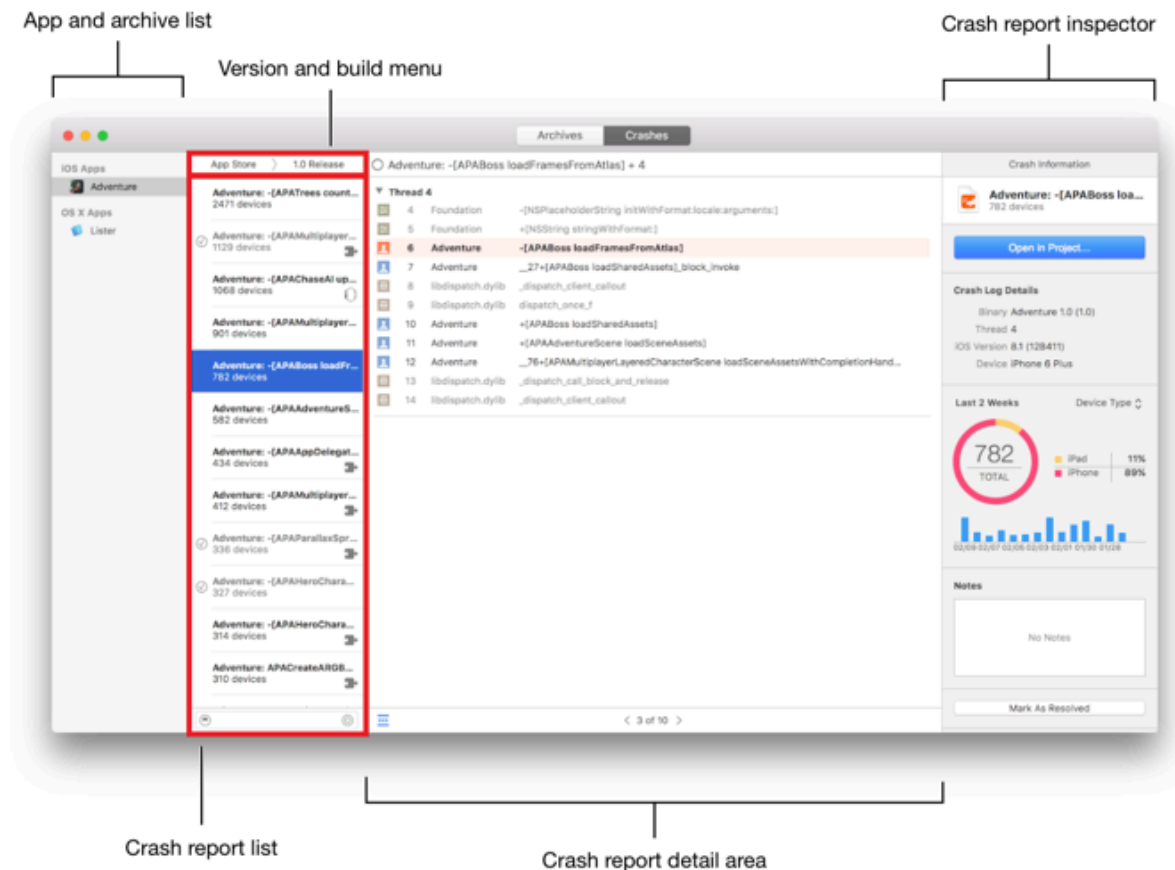
Adding custom logs to the console and to Crash Reports

In iOS 10 there is a lightweight log system framework to add console messages to the app.

Apple provides a service to developers: when a user experiences a crash the system creates a crash log that describes the conditions under which the app terminated, in most cases including a complete stack trace for each executing thread. Then the user can intentionally notify this to Apple and then Apple forwards the crash log to the developer. You, as the developer, can use the log system to insert additional comments and log prints in specific sections of the code to make the debugging easier.

To see the crash reports go to Window->Organizer->Crashes





Alternatively, you can view crash logs directly on the device or import crash reports from a device.

Location-based Services

Core Location

Allows you to find the user location. It was Introduced in iOS 2. You need to import CoreLocation, then use the following main classes

CLLocationManager
 CLLocationManagerDelegate
 CLLocation

Since iOS8 you need to ask the user the permission to track his location. This happens every time the user launches the app. A request for user authorization is displayed automatically when your app first attempts to use location services.

You have two kinds of requests:

- NSLocationAlwaysUsageDescription (works also in background, even with the app closed !! e.g. Uber)
- NSLocationWhenInUseDescription (works only when the app is in use)

Tip: If the user denies consensus don't say "bye bye", because this is a terrible experience for the user. Instead try to provide some service with reduced functionality.

Tip: CL requires a lot of power, thus disable it whenever possible

CL provides the user location (coordinate, altitude, floor, horizontal and vertical accuracy, timestamp, ..)

The location informations are based on

- cell towers (this is the first method the device uses to estimate location)
- Wi-Fi hotspots (every wifi station in the nearby is used to estimate location, even you don't have a password. Consider that every wifi station is actually geolocalized)
- Bluetooth
- GPS (best accuracy but more power consumption)

The Apple Watch may use the iPhone if it is nearby.

If Location Services is on, your iPhone will periodically send the geo-tagged locations of nearby Wi-Fi hotspots and cell towers in an anonymous and encrypted form to Apple, to be used for augmenting this crowd-sourced database of Wi-Fi hotspot and cell tower locations.

iOS starts with the cell tower. If it does not give the requested accuracy it uses wifi APs (ALL OF THEM, even if the device is not connected to them!).

CL provides many services:

- Location updates which gets user location continuously updated
- CL visit (CLVisit) tracks all the places the user visited in a day. Also the arrival and departure times are tracked. It works also in background
- GeoFencing which notifies entry and exit of an area
- Heading (CLHeading object) provides the device orientation with respect to the north pole
- Positioning (CLLocation object)
- Geocoder (CLGeocoder), in the reverse geocoding you pass the location and using a closure it gives you back the street address). You can also use the forward Geocoding to do the opposite
- iBeacon, which advertises the device is approaching a Bluetooth Beacon device

You can activate one or more of those services with the same CLLocationManager

For a given service you can specify a couple of parameters:

- Distance Filter: the minimum distance in meters the user has to move before getting a new update
- Desired Accuracy: You can select it out of a predefined set

Example: Where Am I

1. Let's create an app
2. go to the Storyboard and embed the main VC in a Navigation Controller
3. add the title "Where Am I" to the navigation bar
4. add a label in the VC. Connect it to a IBOutlet
5. add a "Start" BarButton on the navigation bar (right side). Connect it to a IBAction
6. go the VC swift file and import the CoreLocation
7. Create a locationManager class property, but use the lazy version

```
24 lazy var locationManager: CLLocationManager = {
25     let manager = CLLocationManager()
26     manager.delegate = self
27     manager.desiredAccuracy = kCLLocationAccuracyBest
28     manager.distanceFilter = 50
29     return manager
30 }()
```

8. Add the CLLocationManagerDelegate protocol in the VC class definition

```
12 class ViewController: UIViewController, CLLocationManagerDelegate{
```

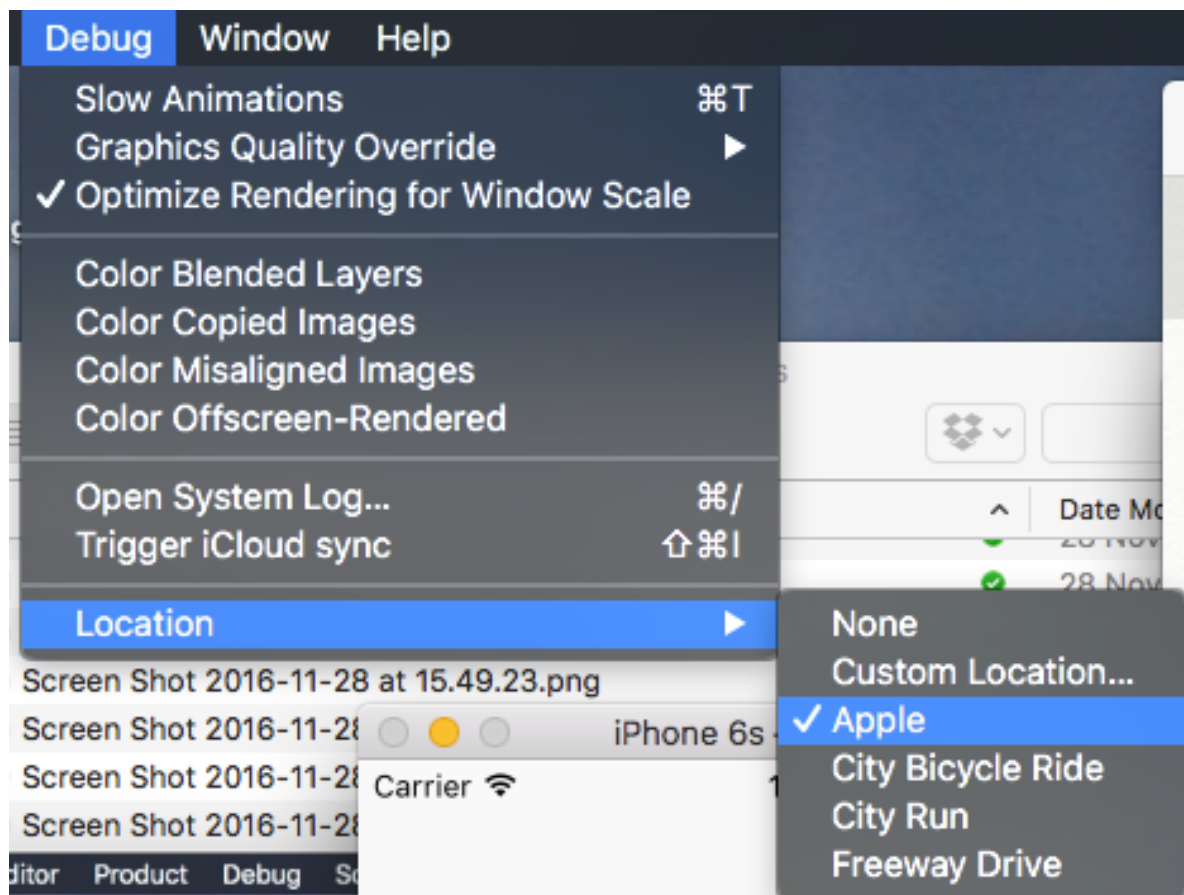
9. in the viewDidLoad get the authorizationStatus and check it

```
37 let status = CLLocationManager.authorizationStatus() //this is a class method
38
39 switch status {
40 case CLAuthorizationStatus.authorizedWhenInUse:
41     print("User has granted authorization (-> enable the start update button)")
42     startLocationButton.isEnabled = true
43 case CLAuthorizationStatus.notDetermined:
44     print("The authorization status is not determined (-> ask the user for authorization)")
45     locationManager.requestWhenInUseAuthorization()
46 default:
47     break
48 }
```

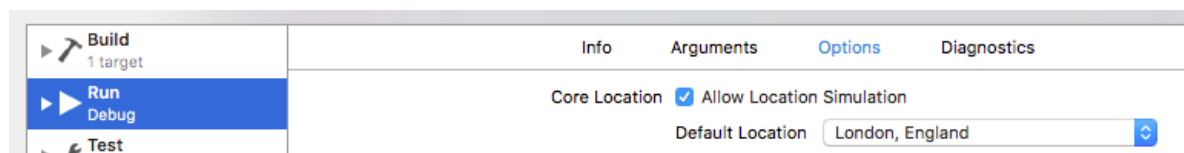
Note: `CLLocationManager.authorizationStatus()` is a class method. So it can be used before an instance of the object has been created. Indeed we have to evaluate it BEFORE to create the location manager object, because we won't do it if we are not authorised to use it.

Note: we use a lazy var for the locationManager property because otherwise self won't exist. This is because self starts to exist after the class init has been executed and this cannot happen before all the properties have been initialised. The lazy var closure will be executed later, at runtime, the first time the property will be used.

In order to debug the app with the simulator you can activate the location utility from the simulator menu



For a more flexible configuration of the simulator, go to the scheme editor, Options



The delegate method which is invoked when new locations are available is

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation])
```

The parameter locations is an array of CLLocation objects in chronological order.

Only the last element is currently used.

```
66 func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
67     guard let loc = locations.last else { return }
68     locationManager.stopUpdatingLocation()
69     geoCoder.reverseGeocodeLocation(loc) { (placemarks: [CLPlacemark]?, error: Error?) in
70         guard error == nil else {
71             fatalError("\(error), \(error?.localizedDescription)") // very useful during development, but don't ship it
72         }
73         guard let pmarks = placemarks, let placemarks = pmarks.first else { return }
74         print("\(placemarks)")
75     }
76 }
```

Note: The first guard is used to unwrap the last element of the locations array. If it is non null we stop the updating activity (to save power), then instantiate a

CLGeocoder object by a class lazy property

```
32     lazy var geoCoder: CLGeocoder = {  
33         return CLGeocoder()  
34     }()
```

Inside the geocoded closure we can test the placemarks array and eventually printout the content.

A placemark is an object of type CLPlacemark, which stores data for a given latitude and longitude. Placemark data includes information such as the country, state, city, and street address,

Piccadilly Circus, Piccadilly Circus, Coventry Street, London, W1J, England @ <+51.51006860,-0.13371130> +/- 100.00m, region CLCircularRegion (identifier:'<+51.51006859,-0.13371130> radius 141.61', center:<+51.51006859,-0.13371130>, radius:141.61m)

Now we are going to print all this stuff.

One of the attributes of a placemark is the addressDictionary, which is declared in this way

```
var addressDictionary: [AnyHashable : Any]? { get }
```

The key type is AnyHashable, which means any type that can generate an hash (e.g. a string).

The value type is Any, which means any type (structure, enumeration, class, ..). This is not standard Swift notation. In Swift an array, for example, is only allowed to have single type elements. An array with Any type elements can have elements of different types. This notation comes from the need to import in Swift an object originated from Obj-C (actually Any is a protocol).

By the way, also exists AnyObject, which means any instance of a class.

In order to read out the addressDictionary we need to check the type.

If we print it in the console we get

```
[  
AnyHashable("Street"): Coventry Street,  
AnyHashable("ZIP"): W1J,  
AnyHashable("Country"): United Kingdom,  
AnyHashable("City"): London,  
AnyHashable("State"): England,  
AnyHashable("Name"): Piccadilly Circus,  
AnyHashable("SubAdministrativeArea"): London,  
AnyHashable("Thoroughfare"): Coventry Street,  
AnyHashable("FormattedAddressLines"): <__NSArrayM  
0x6080000525a0>(  
Piccadilly Circus,  
Coventry Street,
```

```
London,  
W1J,  
England  
)  
AnyHashable("CountryCode"): GB,  
AnyHashable("SubLocality"): Mayfair  
]
```

The entire dictionary is an optional.

Map Kit

for map rendering (2D/3D), do searches, draw the route, fly over it with
MKMapCamera

It was introduced with iOS 3

```
import MapKit  
MKMapView
```

a geographic region is a rectangle with a center and a span

Annotations:

Annotation views are pins to annotate a place.

Create annotation object MKAnnotation (contains the data)

then create an annotation view

In the delegate method mapView() you pass the map view


If you touch a pin you get a collout, a small view composed of 3 parts.

go t the Container app ...


start recording again ..

Add an Annotation

Class:

Subclass of: 

☐ Also create XIB file

Language: 

```

8
9  import MapKit
10
11  class Annotation: NSObject, MKAnnotation {
12
13  }

```

Other

Best practices

Typically developers make their code compatible with the latest iOS version and the version before.

For example, today we would decide to distribute code compatible with iOS 10 and iOS 9.

By the way, iOS 10 is currently adopted by 60% of the customers. 40% of them still stay with iOS 9. This is also due to the fact that you can jailbreak iOS 9, not iOS 10. Other people fear the phone could slow down with the new version. Maintaining compatibility with iOS 9.3 is still possible, but to be compatible with iOS 8 it is a nightmare.

Measurement Unit framework

With iOS 10 Apple made available a new framework to easily handle unit measurement.

see NSMeasurement.

plist file

For each app there is info.plist, a small XML file that iOS reads before the app runs

Hot to stop the simulator from logging like crazy

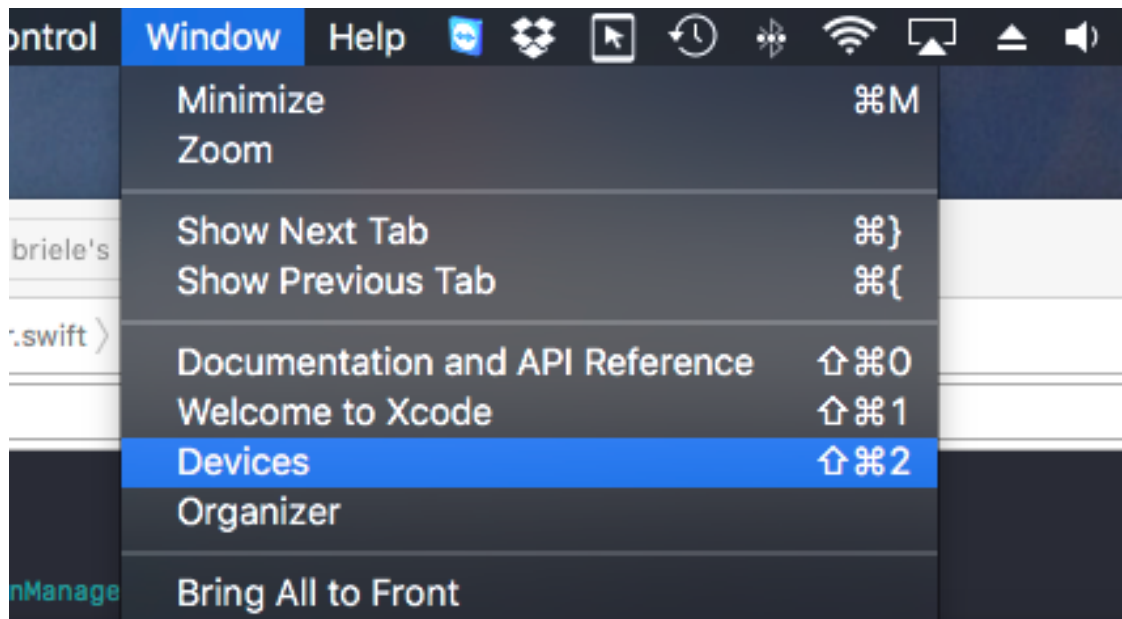
In the Debug scheme, under Arguments section, add the environment variable OS_ACTIVITY_MODE and set it to "disable"

▼ Environment Variables

Name	Value
<input checked="" type="checkbox"/> OS_ACTIVITY_MODE	disable

How to get crash log out from device

To get a crash log from device
connect the device to Xcode, then



This Device			All Logs
Process	Type	Date/Time	
Unknown	CPU Usage	25/10/2016, 00:00	
Unknown	CPU Usage	25/10/2016, 00:00	
Unknown	Unknown		
WhereAmI	Crash	26/10/2016, 13:41	
WhereAmI	Crash	26/10/2016, 13:41	
WhereAmI	Crash	26/10/2016, 12:46	
WhereAmI	Crash	26/10/2016, 12:45	
coreduetd	Crash	26/10/2016, 12:45	

Incident Identifier: 6E26B40C-8D57-45A8-A4C2-2B83B3959377

CrashReporter Key: 70e5d4f9ec53160ced008b53783798c18a44da7f

Hardware Model: iPhone7,2

Process: WhereAmI (216)

Path: /private/var/containers/Bundle/Application/14301E11-7743-4f

WhereAmI.app/WhereAmI

Identifier: com.cosmed.WhereAmI

Version: 1 (1.0)

Code Type: ARM-64 (Native)

Role: Foreground

Parent Process: launchd (1)

Coalition: com.cosmed.WhereAmI (338)

Date/Time: 2016-10-26 13:41:01.9823 +0200

Launch Time: 2016-10-26 13:40:59.9399 +0200

OS Version: iPhone OS 10.0.2 (14A456)

Report Version: 104

Exception Type: EXC_BREAKPOINT (SIGTRAP)

Exception Codes: 0x0000000000000001, 0x000000001001f6848

Termination Signal: Trace/BPT trap: 5

Termination Reason: Namespace SIGNAL, Code 0x5

Terminating Process: exc handler [0]

Triggered by Thread: 0

Filtered syslog:

None found

Thread 0 name: Dispatch queue: com.apple.main-thread

Thread 0 Crashed:

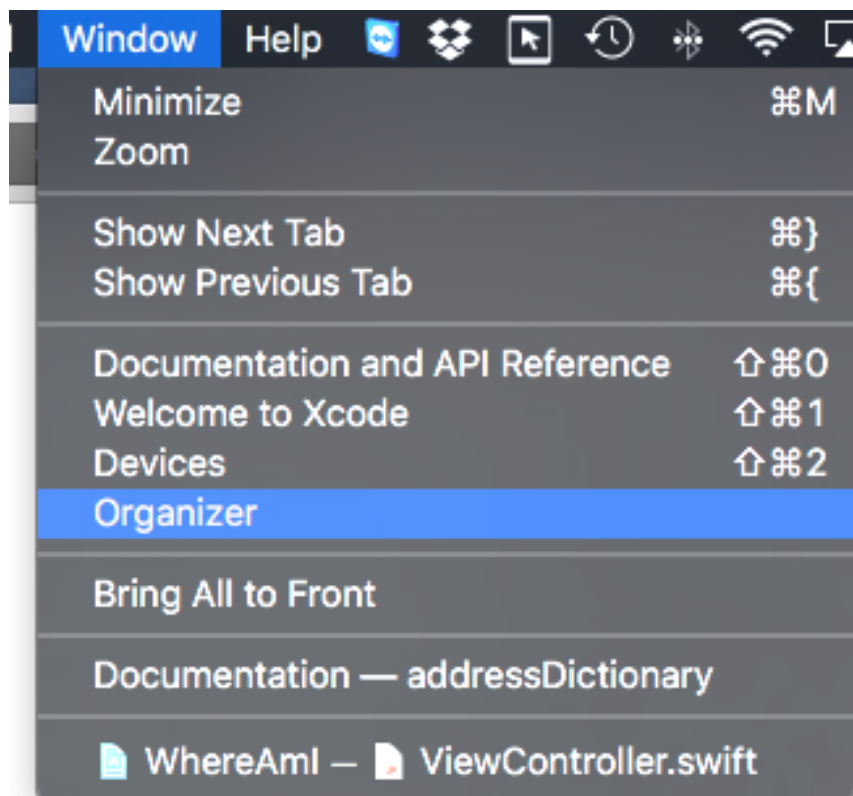
0 libswiftCore.dylib 0x00000001001f6848 0x1001c4000 + 206920

1 libswiftCore.dylib 0x00000001001f6848 0x1001c4000 + 206920

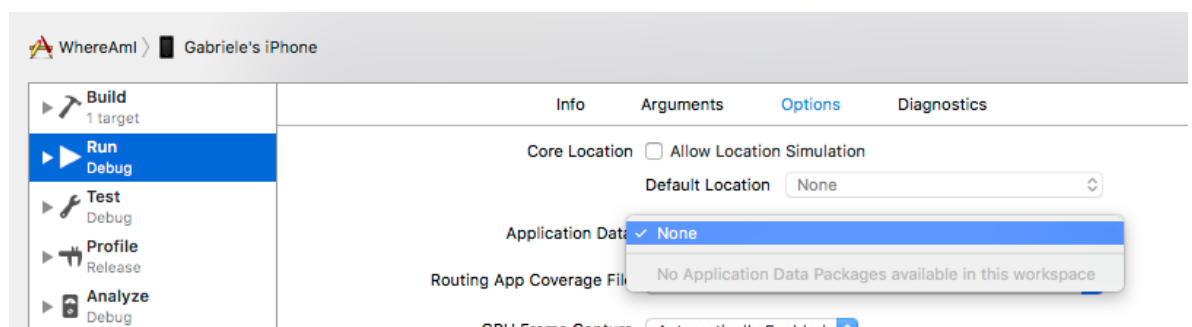
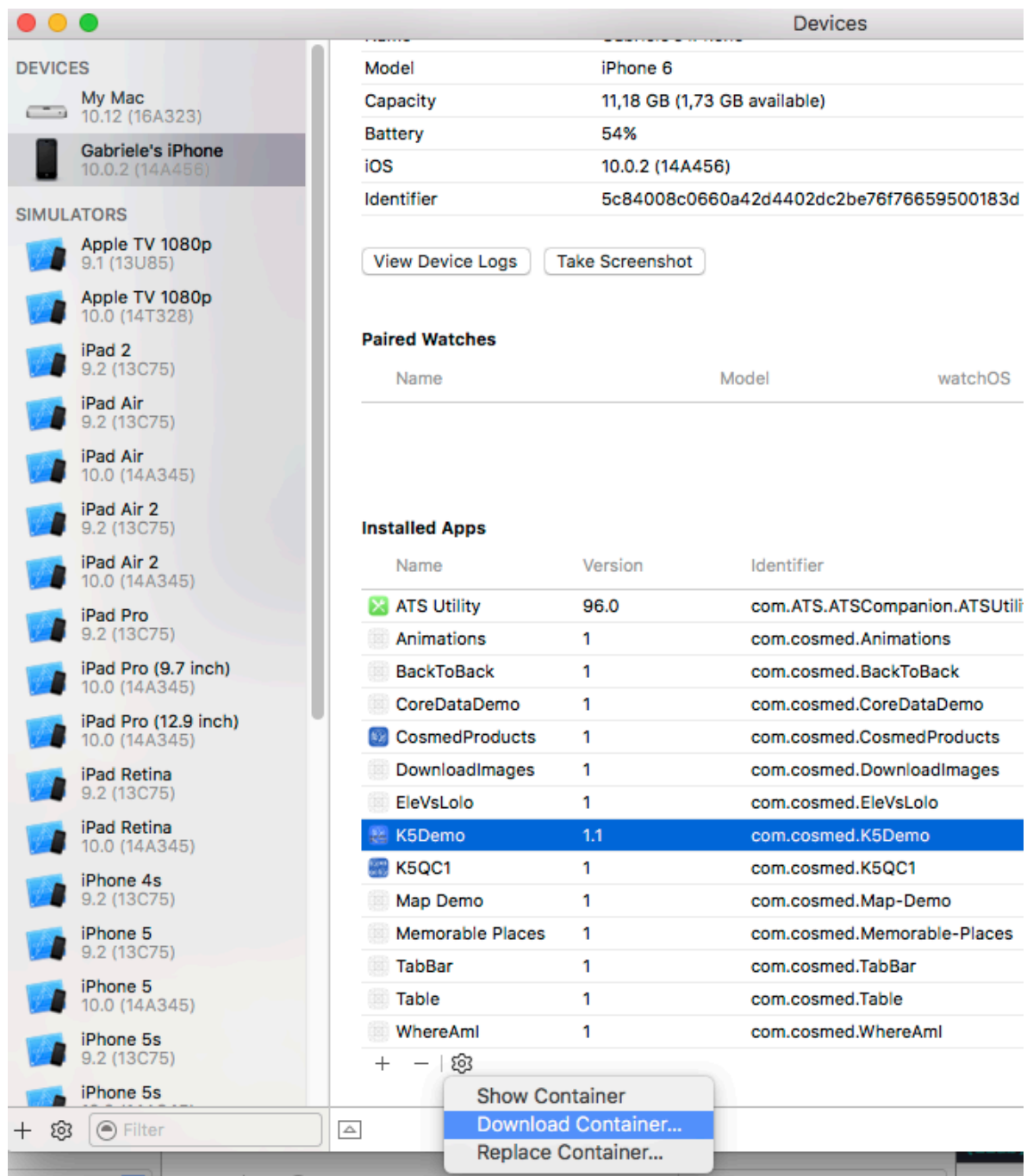
2 WhereAmI 0x00000001000aa540 0x1000a4000 + 25920

TestFlight

TestFlight is an application for developers and it is inside iTunes Connect. If there is a crash on the customer's device then it will be reported in developer's Xcode, under Window->Organizers->Crashes



Spesso non è possibile replicare un crash perché dipendeva dai dati.
Allora il cliente crea il bundle e te lo spedisce. tu lo draghi nel progetto e da
scene->options lo recupero



Tip: BRUTTA IDEA CREARE UN OGGETTO GLOBALE, USA DELEGATION!

Look for Exercise1 and Exercise2 in code..

Extensions

language feature that extend a class with new functionalities, methods and computed properties, .. Example. I take UIColor. Then I add a new method. With needed it with the NotificationCenter class

Notifications

When to use delegation, when notification ?

Delegation is a on-to-one communication. The delegate is only one.

Notification is one-to-many communication. It has the disadvantage is that you don't know whether the object is there and got the notification you send it.

Instead, you can always check if the delegate object exist or not.

Animations

Core Animation

You can run animations and do rendering with the GPU. It was created for the iPhone. It is super-light. Then they port it to mac. Now it is also in the watch.

GPU is the graphic processor. You compile for the CPU

In CA you don't use Views, they are too heavy. CA uses CALayer. It represents a rectangle with a lot of features like center, corner radius, shadow, border, 3D transformations,..

In the CPU you create the CALayer, at runtime it is sent to the GPU via a CATransaction object.

The GPU then generates the presentation layer and a rendered layer which you will see on then screen.

60 FPS is reached in iOS.

When you change a CALayer property it will animate automatically (Implicit animation)

CG: CoreGraphic library. It is used by CA



CA is important because it gives useful feedback to user on app behavior.
Metal gives you more freedom but you have to write a lot of code!

Tip: to comment multiple lines:

1. select the lines
2. Cmd + Shift + "/" (which is Cmd + "//")

You can create a custom animation with CAAAnimation.

There are some subclassess

CABasicAnimation

CAKeyFrameAnimation

CASpringAnimation

CAAnimationGroup

a protocol also exists for all of them

CAMediaTiming

CABasicAnimation

fromValue

toValue

byValue

CAKeyFrameAnimation

CATextLayer to animate text

CAShapeLayer to animate shapes

CAEmitterLayer to create special effect (particle effects, rain, explosion, etc)

read this:

<https://www.invasivecode.com/weblog/replicator-layers-lets-give-a-look-at-the/>

https://www.invasivecode.com/weblog/replicator-layer-core-animation/?doing_wp_cron=1477496393.2041230201721191406250

https://www.invasivecode.com/weblog/caemitterlayer-and-the-ios-particle-system-lets/?doing_wp_cron=1477496496.3232469558715820312500

View Animation

But, if you don't need 3d animation you could work with the view's layer. In iOS 10 we have Interactive Animation that allows to start and stop views animations.

read this:

<https://www.invasivecode.com/weblog/interactive-animation-view-ios-10/>

Layers do not respond to touch screen. Instead with Interactive Animation it is easy to do that. But you need to be careful with the constraints. You can animate also the constraints.

Tip: In a retina resold 1 point = 2 px

Interactive Animations are View Animation, but you can interact with them

Exercise: add the UIAttachBehavior

Core Graphics

For 2D graphics only. It works on the CPU. You allow also to generate and manipulate PDF, images, .. PaintCode is an app that allow you to draw and it generates the code for Core Graphics. Then you create a graphic context and copy the code. There are 3 context: UIView and any subclass of it, PDF, bitmap image. To paint in a view you ask the view its graphic context with `UIGraphicsGetCurrentContext()`

Tip: UIColor is on top of CGColor

Core Text

Good to work with text at the low level

