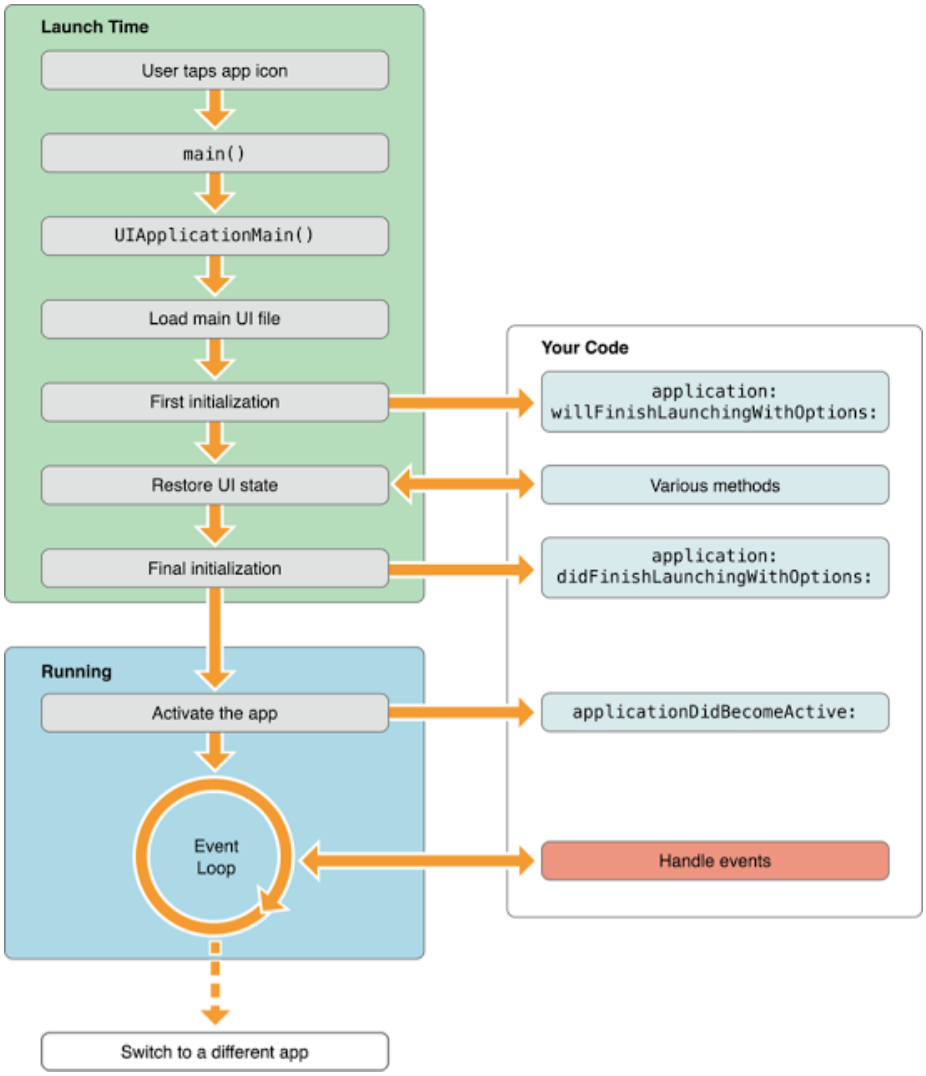
Chapter 2

iOS Development Advance Topics

II.iOS Development Advance Topic

Fig 1.1



1.iOS App Application Life Circle

iOS App Application life circle composes two portions:
1) The App Launch Cycle
2) The View Controller Life Cycle
During the app launch cycle,the iOS system creates a process and main thread for your app and calls your app's main function on that main thread.

The default main function that comes with your Xcode project promptly hands control over to the UIKit framework, it will initialise your app and prepares it to run as demonstrate in the fig 1.1

During View Controller Life Cycle, the view controllers will fire all these events:

- viewDidLoad
- viewWillAppear
- viewDidAppear
- viewWillDisappear
- viewDidDisappear

2. Managing Story Board

1. Segue Events and Methods

Data can be passed between scenes with the method **prepare(for segue: UIStoryboardSegue, sender: Any?)** which is invoked on the view controller when a segue is triggered. This method allows you to customize the setup of the next view controller before it appears on the screen

performSegue(withIdentifier:sender:) to force a transition by calling on the view controller.

3. Advance UI Elements

1. Navigation Controller

A navigation controller manages a stack of view controllers to provide a drill-down interface for hierarchical content. The view hierarchy of a navigation controller is self contained. It is composed of views that the navigation controller manages directly and views that are managed by content view controllers you provide. Each content view controller manages a distinct view hierarchy, and the navigation controller coordinates the navigation between these view hierarchies.

Note: We can reverted back to previous view controller using following methods if view controllers are managed by Navigation Controller

1) **self.dismissViewControllerAnimated //Modal**
2) self.navigationController?.popViewControllerAnimated //Normal
3) self.navigatioinController?.popToRootViewControllerAnimated

Some Useful Methods:
pushViewController:animated: // to push a new view ctrl to the nav stack.
setViewControllers:animated: //To set desired view ctrl upon restored

2. Tabbar Controller

Tabbar organises an app into one or more distinct modes of operation. The view hierarchy of a tab bar controller is self contained. It is composed of views that the tab bar controller manages directly and views that are managed by content view controllers you provide. Each content view controller manages a distinct view hierarchy, and the tab bar controller coordinates the navigation between the view hierarchies

3. Textfield and Its delegation

A UITextField object displays an editable text area in your interface to gather text-based input from the user.

Useful Delegate Methods and Technique:

```
//After ViewController is adapted to UITextFieldDelegate

self.YOURTEXTFILENAME.delegate = self    //Set the Delegate in ViewDidLoad

//use following at suitable place

self.YOURTEXTFILENAME.resignFirstResponder( )       // To hide Keyboard
self.YOURTEXTFILENAME.becomeFirstResponder( )    // To show Keyboard

self.view.endEditing(true)                                    //To hide all keyboard

eg. textFieldShouldReturn delegated method is best place to hide keyboard
func textFieldShouldReturn(_ textField: UITextField) -> Bool {

        textField.resignFirstResponder( )

        return false
}

func textFieldDidEndEditing(_ textField: UITextField) {
        }
```

4. TableView and Its delegation

(see demo : Example of Table view)

To create a table view, several entities in an app must interact: the view controller, the table view itself, and the table view's data source and delegate. The view controller sets the data source and delegate of the table view and sends a

reloadData message to it. The data source must adopt the UITableViewDataSource protocol, and the delegate must adopt the UITableViewDelegate protocol.

Useful Properties and method

YOURTABLEVIEW.indexPathForSelectedRow : used to refer the row selected by user

YOURTABLEVIEW . reloadData :user to reload data

Adding Refresh Control

```
let refreshControl = UIRefreshControl()
refreshControl.addTarget(self, action: #selector(updateNow), for:
UIControlEvents.valueChanged)
tableView.refreshControl = refreshControl
```

Useful Delegate Method and Technique

```
//After ViewController is adapted to UITableViewDataSource, UITableViewDelegate
//and create IBOutlet to TableView

self.YOURTABLEVIEW.delegate = self    //Set the Delegate in ViewDidLoad
self.YOURTABLEVIEW.dataSource = self

//Use following methods———————-
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int)
{
return    0 // NumberofRowAsPerYourData
}

func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
let cell = tableView.dequeueReusableCell(withIdentifier: "YourCellID") as?
YourCustomizedTableviewCellClass
return cell!                          //and return to table
}




func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){

//Perform action when user click such as going to next detail view
//self.performSegueWithIdentifier("segueDetails", sender: self)
}
// set header title for each section if you use more than one section

 func tableView(tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
        if section == 0 {
            return "Section One" //etc… for section ….
        }
  return "Default Title"
}

//You may perform the following segue method before loading subsequent ViewController

 override func prepare(for segue: UIStoryboardSegue, sender: Any?)

//some where else, we can reference destination view controller
let destVc = segue.destination    as! DetailsViewController

}
```

5. Using ScrollView

The UIScrollView class provides support for displaying content that is larger than the size of the application's window. It enables users to scroll within that content by making swiping gestures, and to zoom in and back from portions of the content by making pinching gestures.

Useful Properties and Delegate Methods

YOURSCROLLVIEW.frame -The Outerframe of Scroll View
YOURSCROLLVIEW.contentSize - To set content size
YOURSCROLLVIEW.contentOffset - Where to place content
YOURSCROLLVIEW.minimumZoomScale - Minimum zoomable scale
YOURSCROLLVIEW.maximumZoomScale -Maximum zoomable scale

YOURSCROLLVIEW.addSubView(anotherView) -Adding content view as a child

YOURSCROLLVIEW.autoresizesSubviews = true
YOURSCROLLVIEW.delegate = self

//Delegate Methods

viewForZoomingInScrollView -Asking for a view to scale when the scrollview is zoomed

6. Using Alert View Controller

1. Creating Alert Controller

UIAlertController(**title**: "Title Here", **message**: "Message Here", **preferredStyle**: Style)

Two Styles

UIAlertControllerStyle.alert
UIAlertControllerStyle.actionSheet

2. Adding Action Item

let action1 = UIAlertAction(**title**: "ITEMTITLE", **style**: UIAlertActionStyle.Default, **handler**:{ UIAlertAction in

})

Useful Styles

UIAlertActionStyle.default
UIAlertActionStyle.destructive
UIAlertActionStyle.cancel

3. Presenting Alert Controller

self.present(alertCtrl, animated: true**, completion:** nil**)**

7. Using Picker View

Useful Properties and Delegate Methods

```
//After ViewController is adapted to UIPickerViewDataSource, UIPickerViewDelegate
//and create IBOutlet to PickerView

self.YOURPICKERVIEW.delegate = self        //Set the Delegate in ViewDidLoad
self.YOURPICKERVIEW.dataSource = self

//Use following methods———————-

func numberOfComponents(in pickerView: UIPickerView) -> Int
{
       return 2  //Number of Cols you want
 }

func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {

       if component == 0 {
          return  XX //Number of items you want in col 0

       }
       if component == 1  {
          return  XX //Number of items you want in col 1

       }
       return 0
   }

// title content for row in given column
   func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component:
Int) -> String? {

//User variable component and row to get value for title string
       if component == 0 {
          return "ITEM TEXT IN COLUM 0"
       }

       if component == 1 {
          return  "ITEM TEXT IN COLUM 1"
 }

          return "Invalid Row"
   }

func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
{
//User variable component and row to perform desired action
}
```

4. User Interaction

1. Gesture Types and Detection

Using UIGestureRecognizer, we can intercepts touches that are on their way to being handled by a view. When it recognizes a particular gesture, it calls a method on the object of your choice. There are several types of gesture recognizers built into the SDK.

Notes: ImageView are not able to accept gestures.

-UITapGestureRecognizer

-UIPinchGestureRecognizer

-UISwipeGestureRecognizer

-UIPanGestureRecognizer

-UIRotateGestureRecognizer

-UIScreenEdgeGestureRecognizer

-UILongPressGestureRecognizer

Properties

.numberOfTapsRequired

.delayTouchBegan

.requireGestureRecognizerToFail

.location(in:UIView?)

e.g

```
let tapGesture = UITapGestureRecognizer(target: self, action: "tap:")
tapGesture.numberOfTapsRequired = 2
self.view.addGestureRecognizer(tapGesture)

let pinchGesture = UIPinchGestureRecognizer(target: self, action: "pinch:")
self.view.addGestureRecognizer(pinchGesture)

let swipeGesture = UISwipeGestureRecognizer(target: self, action: "swipe:")
 swipeGesture.direction = UISwipeGestureRecognizerDirection.Down
self.view.addGestureRecognizer(swipeGesture)
```

```
        let rotGesture = UIRotationGestureRecognizer(target: self, action: "rotate:")
        self.view.addGestureRecognizer(rotGesture)

        let panGesture = UIPanGestureRecognizer(target: self, action: "pan:")
         self.view.addGestureRecognizer(panGesture)

        let edgeGesture = UIScreenEdgePanGestureRecognizer(target: self, action:
"screenEdge:")
        self.view.addGestureRecognizer(edgeGesture)

        let lpGesture = UILongPressGestureRecognizer(target: self, action:
"longPress:")
        self.view.addGestureRecognizer(lpGesture)

//Example Events to intercept
func tap(gesture: UIGestureRecognizer) {
         print(gesture.locationInView(self.view))

}

func longPress(gesture: UIGestureRecognizer) {

         //Load and create Menu
         let menu = UIMenuController.sharedMenuController()

        let menuitem1 = UIMenuItem(title: "Help", action: "help:")
        let menuitem2 = UIMenuItem(title: "Delete", action: "del:")

        menu.menuItems = [ menuitem1,menuitem2]

        menu.setTargetRect(CGRect(x:gesture.locationInView(self.view).x,
y:gesture.locationInView(self.view).y, width:2, height:2),
          inView: self.view)
        menu.setMenuVisible(true, animated: true)

}

//To load menuController, this function is required
override func canBecomeFirstResponder() -> Bool {
        return true
    }
```

2. Using TouchID

Sample Code:

```
import LocalAuthentication

let context = LAContext()
var error: NSError?

// check if Touch ID is available
 do {
try  context.canEvaluatePolicy(.DeviceOwnerAuthenticationWithBiometrics, error: nil)
let reason = "Authenticate with Touch ID"
context.evaluatePolicy(.DeviceOwnerAuthenticationWithBiometrics, localizedReason: reason, reply:
 {(succes: Bool, error: NSError?) in
  if succes {
    print("Touch ID Authentication Succeeded")
  }
  else {
   print("Touch ID Authentication Failed")
  }
})
} catch var error1 as NSError {
    error = error1
    print("Touch ID Not Available")
}
```

3. Notification

Local and push notifications, called remote notification, are great for keeping users informed with timely and relevant content, whether your app is running in the background or inactive. Notifications can display a message, play a distinctive sound, or update a badge on your app icon and our app may take different actions, such as downloading new data.

-**Local Notification**: scheduled and sent by the app itself, without necessary involvement of the Internet.

-**Push Notification**: arrived from outside the device and is pushed to your app , originated from your remote server , on a user's device via the Apple Push Notification service ,(APNs).

-App level Notification Using NSNotificationCenter to trigger events

1. Local Notification

Step1 : Registering Local Notification

```
//In AppDelegate

let  notiType:UIUserNotificationType =  [UIUserNotificationType.Alert ,
UIUserNotificationType.Sound , UIUserNotificationType.Badge]

 let notiSetting = UIUserNotificationSettings(forTypes: notiType,
categories: nil)
 UIApplication.sharedApplication().
 registerUserNotificationSettings(notiSetting)

func application(application: UIApplication,
didRegisterUserNotificationSettings notificationSettings:
UIUserNotificationSettings) {
print("Registered Local Noti") }
```

Step2 : Scheduling Local Notification

```
//Use in ViewController
let noti1 = UILocalNotification()

 noti1.fireDate = NSDate(timeIntervalSinceNow: 2)//Next two minute
 noti1.alertTitle = "title....."
 noti1.alertBody = "alert body"
noti1.soundName =  UILocalNotificationDefaultSoundName
 noti1.applicationIconBadgeNumber = 1
 UIApplication.sharedApplication().scheduleLocalNotification(noti1)
```

Step3 : Responding Local Notification

```
//Only respond when app is in foreground, used in AppDelegate

func application(application: UIApplication, didReceiveLocalNotification
notification: UILocalNotification) {
UIApplication.sharedApplication().applicationIconBadgeNumber =
UIApplication.sharedApplication().applicationIconBadgeNumber - 1 }
```

Note:
Actionable notifications let you add custom action buttons to the standard iOS interfaces for local and push notifications. Actionable notifications give the user a quick and easy way to perform relevant tasks in response to a notification. Prior to iOS 8, user notifications had only one default action. In iOS 8 and later, the lock screen, notification banners, and notification entries in Notification Center can display one or two custom actions. Modal alerts can display up to four. When the user selects a custom action, iOS notifies your app so that you can perform the task associated with that action.

2. App Level Notification

Step1 : Registering Events

```
NotificationCenter.default.addObserver(self,
selector: #selector(YourFunctiontoExecute),
 name:NSNotification.Name(rawValue:"AwaitingEventName"),
object: nil)


func YourFunction(_ notification: Notification) {
        print("Yes i know the event, NameOfEvent, is triggered");
}
```

Step2 : Firing Events in Anywhere in App

```
NotificationCenter.default.post
(name:Notification.Name(rawValue:"EventName") ,          object: nil)

//Will load eventFunction registered to EventName in step 1
```

5. Multimedia

1. Playing Sound and Video

A. Useful Methods for Sound

```
//setup

var player: AVAudioPlayer = AVAudioPlayer()

or

let path  = Bundle.main.path(forResource: "01adele", ofType: "mp3")
let musicFileURL = URL(fileURLWithPath: musicFileLocation!)
player = AVPlayer(url: musicFileURL)


player.play()
player.pause()
```

B. Useful Properties for Sound

```
player.currentItem!.duration.seconds          -get the song length
player.currentTime().second                   -Set the location of Song
player.volume                                 -Set the volume

let cmt = CMTime(seconds:time, preferredTimescale: 1)
player.seek(to: cmt)
```

C. Useful Methods and Technique for Video

```
//setup

let moviePath = NSBundle.mainBundle().pathForResource("movie", ofType:
"mp4")!
let url = NSURL.fileURLWithPath(moviePath)
let player = AVPlayer(URL: url)

let playerViewController = AVPlayerViewController()
playerViewController.player = player

self.addChildViewController(playerViewController)
playerViewController.view.frame = CGRectMake(20, 50, 300, 300)
//playerViewController.view.frame = self.view.frame
self.view.addSubview(playerViewController.view)

player.play()
```

2. Using Timer

A. Setting up

```
timer = Timer.scheduledTimerWithTimeInterval(1.0, target: self, selector:
"triggerEvent", userInfo: nil, repeats: true)
timer.fire( ) //Start the timer

func triggerEvent( )
```

```
{
        //To Do
}
```

B. Methods

timer.invalidate() - To Stop Timer

3. Using Camera And Photo Library

Using UIImagePickerController, we can pickup video and photo from alblum and shoot ourself and store it in library. By incorporating protocols, UIImagePIckerControllerDelegate and UINavigationControllerDelegate, UIImagePickerController will perform transition and return the information as follow:

A. Setting up

```
import MobileCoreServices //for constant KUTType
//Type?     --> Image, Movie
let imageMediaType = kUTTypeImage as NSString as String
let movieMediaType = kUTTypeMovie as NSString as String

//In Desired Triggered Action to pickup image or shoot

let imagePicker = UIImagePickerController()
imagePicker.delegate = self
imagePicker.sourceType = UIImagePickerControllerSourceType.Camera
imagePicker.mediaTypes =
UIImagePickerController.availableMediaTypesForSourceType(source)!
imagePicker.allowsEditing = true
self.presentViewController(imagePicker, animated: true,
                completion: nil)
```

B. Getting Data

```
//Delegate Methods
func imagePickerController(picker: UIImagePickerController,
            didFinishPickingMediaWithInfo info: [String : AnyObject]) {

 let mediaType = info[UIImagePickerControllerMediaType] as! String
  self.dismissViewControllerAnimated(true, completion: nil)

switch (mediaType)
 {
 case imageMediaType:
        let image = info[UIImagePickerControllerOriginalImage]  as! UIImage
        //Now image can be user for ImageView or save it
        UIImageWriteToSavedPhotosAlbum(image,
        self,"image:didFinishSavingWithError:contextInfo:", nil)


   case movieMediaType:
            let pickedVideo =  info[UIImagePickerControllerMediaURL] as! NSURL
            let selectorToCall = Selector("video:didFinishSavingWithError:context:")
            if (newMedia == true)
            {
```

```
            UISaveVideoAtPathToSavedPhotosAlbum(pickedVideo.relativePath!, self,
        selectorToCall, nil)

            //Save to Documents if needed
            // Save the video to the app directory so we can play it later
              let videoData = NSData(contentsOfURL: pickedVideo)


let documentsPath = NSSearchPathForDirectoriesInDomains(.DocumentDirectory,
                    .UserDomainMask, true)[0]

let dataPath =
              documentsPath.stringByAppendingPathComponent("testFile.mp4")
              videoData?.writeToFile(dataPath, atomically: false)
              self.dismissViewControllerAnimated(true, completion: nil)
              }
         default: print("invalid mediatype")

      }

func image(image: UIImage, didFinishSavingWithError error: NSErrorPointer,
contextInfo:UnsafePointer<Void>) {

            if error != nil {
               postAlert( "Error",message: "Saving Failed")
            }
            else
            {
               postAlert( "Success",message: "Image Saving Succeed")
            }
}

func video( video:String    , didFinishSavingWithError error: NSError!,
context:UnsafeMutablePointer<Void>){
      if error != nil {
         postAlert( "Error",message: "Saving Failed")
      }
      else
      {
         postAlert( "Success",message: "Vidoe Saving to Album Succeed")
      }
   }

// MARK: Utility methods for app
   // Utility method to display an alert to the user.
   func postAlert(title: String, message: String) {
      let alert = UIAlertController(title: title, message: message,
         preferredStyle: UIAlertControllerStyle.Alert)
      alert.addAction(UIAlertAction(title: "Ok", style: UIAlertActionStyle.Default,
handler: nil))
      self.presentViewController(alert, animated: true, completion: nil)
   }
```

6. Swift Programming Part II

1. Class

I. Introduction

In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behaviour (member functions or methods).Classes are composed from structural and behavioural constituents.
The structure defined by the class determines the layout of the memory used by its instances.The behaviour of class or its instances is defined using methods. Methods are subroutines with the ability to operate on objects or classes. Some types of methods are created and called by programmer code, while other special methods —such as constructors, destructors, and conversion operators—are created and called by compiler-generated code. A language may also allow the programmer to define and call these special methods.

eg.
```
class <classname> {
Definition 1
Definition 2
---
Definition N
}
```

eg.
```
class student{
var studname: String = "Alex"
var mark: Int = 80
var mark2: Int = 76
}
```

eg.
```
class Shape {
   var numberOfSides = 0

   func simpleDescription() -> String {
      return "A shape with \(numberOfSides) sides."
   }

}
```

The syntax for creating instances

```
let stuRecord = student()
var triangle = Shape( )
```

And access the properties and methods as of objects as follow:

stuRecord.studname

```
triangle.numberofSides = 3
triangle.simpleDescription( )
```

II.Class with Initialiser

First version of the Shape class is missing something important: an initializer to set up the class when an instance is created. Use init to create one.

```
class NamedShape {
    var numberOfSides: Int = 0
    var name: String

    init(name: String) {
        self.name = name //****
    }

    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

To Create Instance:

```
var shape = NamedShape("square")
print(shape.name)
```

Note:
self is used to distinguish the name property from the name argument to the initializer. The arguments to the initializer are passed like a function call when you create an instance of the class. Every property needs a value assigned—either in its declaration (as with numberOfSides) or in the initializer (as with name).And also Use deinit to create a deinitializer if you need to perform some cleanup before the object is deallocated.

III.Sub Classing and Inheritance:

The act of basing a new class on an existing class is defined as 'Subclass'. The subclass inherits the properties, methods and functions of its base class. To define a subclass ':' is used before the base class name.

eg.

```
class Square: NamedShape { // Now Inheritance from class NamedShape
    var sideLength: Double          //New variable, Not in Super Class

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 4
    }

    func area() ->  Double {
        return sideLength * sideLength
    }

    override func simpleDescription() -> String {
        return "A square with sides of length \(sideLength)."
    }
```

```
}
let test = Square(sideLength: 5.2, name: "my test square")
test.area()
test.simpleDescription()”
```

Note:
'super' keyword is used as a prefix to access the methods, properties and subscripts declared in the super class. Subclass provides the concept of overriding. 'override' keyword is used to override the methods declared in the superclass. And 'final' keyword is used with properties and method in the parent class to prevent overriding throughout the child hierarchy

Classes in Swift refers multiple constants and variables pointing to a single instance. To know about the constants and variables pointing to a particular class instance identity operators are used. Class instances are always passed by reference. In Classes NSString, NSArray, and NSDictionary instances are always assigned and passed around as a reference to an existing instance, rather than as a copy.

eg.

Identical to Operators	Not Identical to Operators
Operator used is (===)	Operator used is (!==)

2. Struct & Enumeration

Use struct to create a structure. Structures support many of the same behaviors as classes, including methods and initializers. One of the most important differences between structures and classes is that structures are always copied when they are passed around in your code, but classes are passed by reference.

Use enum to create an enumeration. Like classes and all other named types, enumerations can have methods associated with them.

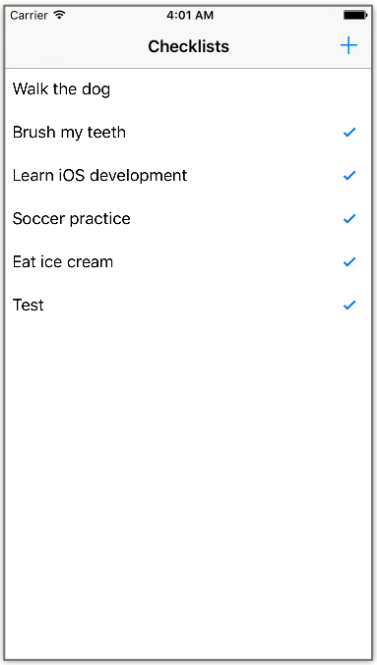
e.g

```
enum Rank: Int {
   case Ace = 1
   case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
   case Jack, Queen, King
   func simpleDescription() -> String {
      switch self {
      case .Ace:
         return "ace"
      case .Jack:
         return "jack"
      case .Queen:
         return "queen"
      case .King:
         return "king"
      default:
         return String(self.rawValue)
      }
   }
}
```

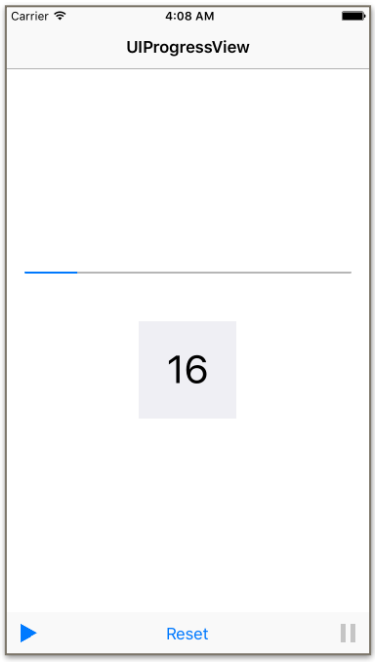
```
let ace = Rank.Ace
let aceRawValue = ace.rawValue
```

2. Exercise

1. Create a projects as shown in pic.1 using tableView



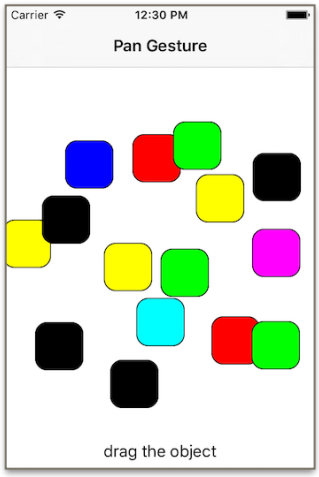
(pic.1)



(pic.2)

2. Create an app as shown in pic.2 using ProgressView and Timer, which let the progress view to fill up to 100% while counting the digit

3. Create an app that notify you at the designated time using Date and time picker and local notification.



(pic.3)

4. Create an app show in pic3 wherein the objects are able to move by a touch

5. Create an app that takes one photo from user album and the photo are later able to rotate , pinch for zooming effect.

6. Create a simple Music Player able to play designated songs from a List.

7. Type the following class definition and create additional two rectangles. rect2 and rect3

```
class rectangle {
var length: Double
var breadth: Double
init() {
length = 6
breadth = 12
}
}
var rect1 = rectangle()
print(rect1.length)
```

8. In previous example, modify the initialiser of class to accept length and breadth as a parameters. And put a function, able to compute an area of that rectangle and display.

9. Create a square class that inheritance from rectangle class in ex.1.But the propertie 'breadth' stored value will be initialise as the value in length properties

10. Create Two Initialiser in Class rectangle in exercise, one requiring with parameter to initialise length and breadth and other with default value as 12 for each properties.

11. Create a square objects, named square1, with its length as 5 unit. Assign new square object newSquare by square1. Print out both length. Change the original square square1 length to 7 and observe the newSquare's length again.

12. Create one array of square objects storing square1 and newSquare. and also create empty array to store square objects for future used.