

UNIT 8

Introduction to iOS Programming

(18 Hours) — Mobile Programming (CAC351)

8.1 Introduction to iOS and iPadOS Programming

iOS is Apple's proprietary mobile operating system that runs on iPhone (and iPod touch). **iPadOS** is the variant of iOS for iPad, adding multitasking, split view and Apple Pencil features. Apple also has watchOS, tvOS and visionOS. iOS was first released in 2007 as iPhone OS (renamed iOS in 2010). **iOS programming** means developing apps for these devices using Apple's tools, mainly the **Swift** language, the **Xcode** IDE and the **UIKit** or **SwiftUI** frameworks.

Note: The syllabus text reads "ios and los programming"; this appears to be a typing error for "iOS and iPadOS programming" (the 'l' stands for 'i'), and this unit is written on that basis.

Features of iOS

- Closed-source, Unix-based (Darwin/XNU kernel); strong security (sandboxing, code signing, Face ID/Touch ID).
- Consistent hardware and software ecosystem – fewer device variations than Android.
- Regular simultaneous OS updates for all supported devices.
- Apps are distributed through the Apple App Store after App Review.
- Design follows Apple's Human Interface Guidelines (HIG).

Version History (Selected)

Year	Version	Highlights
2007	iPhone OS 1	First iPhone; multi-touch UI
2008	iPhone OS 2	App Store and third-party apps
2010	iOS 4	Named 'iOS'; multitasking
2013	iOS 7	Flat redesign
2014	iOS 8 / Swift	Swift language introduced
2019	iOS 13 / iPadOS	iPadOS split; SwiftUI announced; dark mode

Year	Version	Highlights
2020–2024	iOS 14 to 18	Widgets, Focus, Live Activities, Apple Intelligence (iOS 18)

8.2 iOS Platform

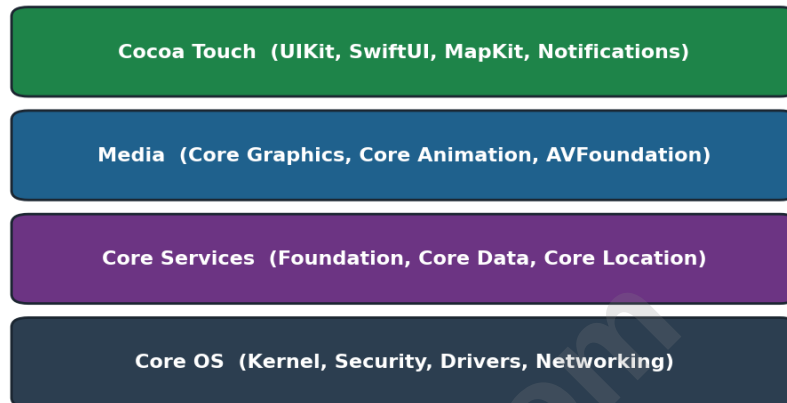


Figure 8.1: The four layers of the iOS architecture

Layer	Contents
Cocoa Touch	UIKit (UI, touch, views), SwiftUI, notifications, multitasking, MapKit, gesture recognisers
Media	Core Graphics, Core Animation, AVFoundation (audio/video), Metal, image and text drawing
Core Services	Foundation, Core Data, Core Location, SQLite, networking (URLSession), iCloud
Core OS	Kernel (XNU), memory, security services, low-level networking, drivers

Design Pattern: MVC

UIKit apps follow **Model–View–Controller**: the **Model** holds data and logic, the **View** displays data and captures input, and the **Controller** (a *UIViewController*) connects the two.

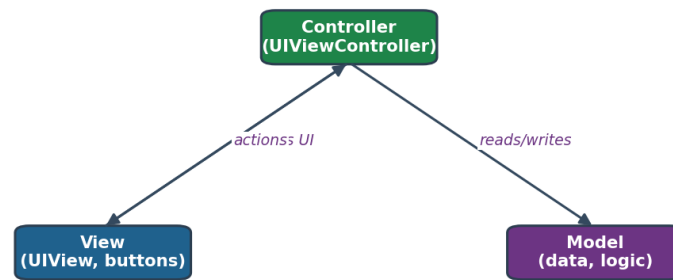


Figure 8.2: MVC in iOS

iOS vs Android

Aspect	iOS	Android
Developer	Apple	Google
Language	Swift, Objective-C	Kotlin, Java
IDE	Xcode (macOS only)	Android Studio (Win/Mac/Linux)
UI definition	Storyboard / SwiftUI / code	XML layouts / Compose
Store	App Store (strict review)	Google Play and others
Open source	No	Mostly (AOSP)
Devices	Apple hardware only	Many manufacturers
Developer fee	US\$99 per year	US\$25 one time

8.3 Environment Setup

- A Mac running a supported macOS version is required (Xcode does not run on Windows/Linux).
- Install **Xcode** free from the Mac App Store (large download, includes iOS SDK, Simulator, Interface Builder, Instruments).
- Open Xcode and accept the licence; install additional components and simulator runtimes via **Xcode → Settings → Components/Platforms**.
- Sign in with an **Apple ID** in *Settings → Accounts* – a free account lets you run apps on your own device; a paid **Apple Developer Program** membership (US\$99/year) is needed to publish to the App Store and use some services.
- Optional: connect an iPhone by cable, enable **Developer Mode** on it (Settings → Privacy & Security) and trust the computer.

Term	Meaning
Xcode	Apple's IDE: code editor, Interface Builder, debugger, simulator
Simulator	Virtual iPhone/iPad running on the Mac
Interface Builder	Visual editor for storyboards and XIB files
Storyboard	File that shows screens (scenes) and the flow (segues) between them
Provisioning profile / certificate	Code-signing files that allow installing on real devices
Info.plist	Configuration file: app name, permissions descriptions, orientations

8.4 Creating an Xcode Project

- Open Xcode → **Create New Project** (File → New → Project).
- Choose **iOS → App** template and click Next.
- Enter *Product Name* (e.g., HelloiOS), *Team*, *Organization Identifier* (com.example) – together they form the *Bundle Identifier*.
- Select *Interface: Storyboard* (UIKit) or SwiftUI, and *Language: Swift*.
- Choose a save location and click Create.

Project file	Role
AppDelegate.swift / SceneDelegate.swift	App and scene life-cycle events (launch, background, foreground)
ViewController.swift	Code for the first screen
Main.storyboard	Visual layout of screens
LaunchScreen.storyboard	Splash screen shown while app starts
Assets.xcassets	Images, colours and app icon
Info.plist	App configuration and permission descriptions

8.5 Introduction to the Swift Language

Swift is a modern, safe, fast, open-source programming language created by Apple (2014) for iOS, macOS, watchOS and tvOS. It replaces Objective-C in most new projects. Key features: type safety and type inference, **optionals** for handling missing values, closures, structs and enums with methods, protocols, automatic memory management (ARC) and no semicolons required.

Variables, Constants and Types

```
let pi = 3.14159           // constant (cannot change)
var score = 0              // variable (can change)
var name: String = "Asha"  // explicit type annotation
let isPass: Bool = true
let age: Int = 21
let price: Double = 99.5

score += 10
print("Hello, \(name)! Score: \(score)") // string interpolation
```

Listing 8.1: Basics

Control Flow

```
if score >= 40 {
    print("Pass")
} else if score >= 30 {
    print("Grace")
} else {
    print("Fail")
}

for i in 1...5 { print(i) } // closed range 1 to 5
for city in ["KTM", "PKR"] { print(city) }
var n = 3
while n > 0 { n -= 1 }

switch score {
case 0..<40:    print("Low")
case 40..<80:   print("Medium")
default:      print("High")
}
```

Listing 8.2: if, loops and switch

Collections

```
var fruits = ["Apple", "Mango"] // Array
fruits.append("Banana")
let marks = ["Math": 90, "Java": 85] // Dictionary
let unique: Set = [1, 2, 3] // Set
print(fruits.count, marks["Math"] ?? 0)
```

Listing 8.3: Array, Dictionary and Set

Functions, Optionals and Closures

```

func greet(_ person: String, from city: String = "Kathmandu") -> String {
    return "Hello \(person) from \(city)"
}
print(greet("Ram"))

var nickname: String? = nil           // optional - may hold no value
if let nick = nickname { print(nick) } // optional binding
let safe = nickname ?? "No nickname"  // nil-coalescing
// nickname! forces unwrap - crashes if nil, avoid it

let double = { (x: Int) -> Int in x * 2 } // closure
let evens = [1, 2, 3, 4].filter { $0 % 2 == 0 }

```

Listing 8.4: Functions, optionals, closures

Classes, Structs and Protocols

```

struct Point { var x: Int; var y: Int }           // value type (copied)

class Student {                                   // reference type
    var name: String
    var roll: Int
    init(name: String, roll: Int) { self.name = name; self.roll = roll }
    func describe() -> String { "\(name) (\(roll))" }
}
class GradStudent: Student {                     // inheritance
    override func describe() -> String { "Graduate: " + super.describe() }
}
protocol Payable { func pay() -> Double }        // protocol (like interface)

```

Listing 8.5: Struct vs class

Struct	Class
Value type – copied on assignment	Reference type – shared reference
No inheritance	Supports inheritance
Memberwise initialiser automatically	Must write an init
Preferred for simple data	Used for UIViewController, shared state

8.6 Views and the View Hierarchy

A **view** (*UIView*) is a rectangular area on screen that draws content and handles touches. *UILabel*, *UIButton*, *UITextField* and *UIImageView* are subclasses. Views are nested: each view has a **superview** and zero or more **subviews**, forming the **view hierarchy** with a *UIWindow* at the top and the view controller's root *view* below. Subviews are drawn on top of their superview, and a view's position is described by its **frame** (relative to superview) and **bounds** (own coordinate space).

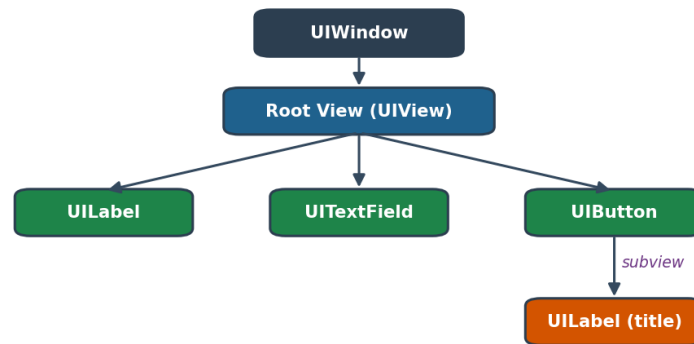


Figure 8.3: Example iOS view hierarchy

Auto Layout

Auto Layout positions views using **constraints** (leading, trailing, top, bottom, width, height, centre) so the interface adapts to any screen size and orientation. Use the *Add New Constraints* button in Interface Builder, and respect the **Safe Area** to avoid the notch and home indicator. **UIStackView** arranges views automatically in a horizontal or vertical stack.

```
// Programmatic view and constraints
let label = UILabel()
label.text = "Hello iOS"
label.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(label)
NSLayoutConstraint.activate([
    label.centerXAnchor.constraint(equalTo: view.centerXAnchor),
    label.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 40)
])
```

Listing 8.6: Adding a view with Auto Layout constraints in code

8.7 Storyboard and View Controllers

A **Storyboard** is a visual representation of an app's screens. Each screen is a **scene** controlled by a **view controller** (*UIViewController*). Scenes are connected by **segues** (transitions); the arrow entering a scene marks the **Initial View Controller**.

View Controller Life Cycle

Method	Called when
<code>viewDidLoad()</code>	View loaded into memory (once) – set up UI and data
<code>viewWillAppear(_:)</code>	Just before the view appears (every time)
<code>viewDidAppear(_:)</code>	View is on screen – start animations, network calls
<code>viewWillDisappear(_:)</code>	Just before the view is removed or covered

Method	Called when
viewDidDisappear(_:)	View is no longer visible – stop tasks

Navigation Types

- **UINavigationController** – push/pop stack of screens with a back button.
- **UITabBarController** – tabs at the bottom.
- **Segue types** – Show (push), Present Modally, Popover; data passed in *prepare(for:sender:)*.
- Programmatic: *present(vc, animated: true)*, *navigationController?.pushViewController(vc, animated: true)*.

```
// Passing data with a segue (Identifier "showDetail" set in Storyboard)
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetail",
        let detail = segue.destination as? DetailViewController {
        detail.studentName = nameField.text ?? ""
    }
}
```

Listing 8.7: *prepare(for:sender:)*

Note: Compare with Android: ViewController ≈ Activity; Segue ≈ Intent; Storyboard ≈ navigation graph; Auto Layout ≈ ConstraintLayout.

8.8 Building the Interface and Making Connections

Building the Interface

- Open **Main.storyboard**.
- Press the **+** (Library) button and drag widgets (Label, Button, Text Field, Image View) onto the view controller.
- Set text, colours and fonts in the **Attributes inspector** (right pane).
- Add Auto Layout constraints; check *View as* (iPhone SE to iPad) at the bottom to preview sizes.

Making Connections: Outlets and Actions

Interface Builder objects are linked to Swift code by two kinds of connections:

Connection	Direction	Purpose	Declared as
Outlet	UI → code	Reference to a view so code can read/change it	@IBOutlet weak var nameField: UITextField!

Connection	Direction	Purpose	Declared as
Action	UI event → code	Method invoked when an event happens (Touch Up Inside)	@IBAction func tapped(_ sender: UIButton)

How to connect: open the Assistant Editor (storyboard on one side, ViewController.swift on the other), hold **Control** and drag from the widget into the code; choose *Outlet* or *Action*, name it and press Connect. A filled circle beside the code line shows the connection is active. A broken connection (e.g., a deleted outlet name) crashes the app at launch with “*this class is not key value coding-compliant*” – remove the stale connection in the Connections inspector.

8.9 Working with Widgets and Their Attributes

Widget	Class	Common attributes / properties
Label	UILabel	text, font, textColor, textAlignment, numberOfLines
Button	UIButton	title, setTitle(_:for:), titleColor, isEnabled, image
Text Field	UITextField	text, placeholder, keyboardType, isSecureTextEntry, delegate
Text View	UITextView	multi-line editable text
Image View	UIImageView	image,.contentMode (scaleAspectFit / Fill)
Switch	UISwitch	isOn, valueChanged action
Slider / Stepper	UISlider / UIStepper	value, minimumValue, maximumValue
Segmented Control	UISegmentedControl	selectedSegmentIndex
Picker / Date Picker	UIPickerView / UIDatePicker	date, datePickerMode
Table View / Collection View	UITableView / UICollectionView	Lists / grids (like RecyclerView) via dataSource and delegate
Activity Indicator	UIActivityIndicatorView	startAnimating(), stopAnimating()

```

class WidgetViewController: UIViewController {
    @IBOutlet weak var nameField: UITextField!
    @IBOutlet weak var resultLabel: UILabel!
    @IBOutlet weak var newsSwitch: UISwitch!

    override func viewDidLoad() {
        super.viewDidLoad()
        nameField.placeholder = "Enter your name"
        resultLabel.numberOfLines = 0
    }
    @IBAction func showTapped(_ sender: UIButton) {
        let name = nameField.text ?? ""
        resultLabel.text = newsSwitch.isOn ? "Hi \(name), thanks for subscribing"
                                                : "Hi \(name)"
        view.endEditing(true)           // hide keyboard
    }
    @IBAction func alertTapped(_ sender: UIButton) {
        let alert = UIAlertController(title: "Saved", message: "Data saved",
                                     preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default))
        present(alert, animated: true)
    }
}

```

Listing 8.8: Outlets, actions and an alert

8.10 Creating a Simple iOS Application

Example — Tip Calculator

Build a screen with a text field for the bill amount, a slider for tip percentage, a label showing the tip and total, and a Calculate button.

Solution steps

1. Create a new iOS App project (Storyboard, Swift) named TipCalc.
2. In Main.storyboard add: UITextField (keyboard Decimal Pad), UISlider (0–30), two UILabels, UIButton; add constraints or embed them in a vertical UIStackView.
3. Create outlets: *billField*, *tipSlider*, *tipLabel*, *totalLabel*; create actions *sliderChanged* and *calculateTapped*.
4. Add the code below, then choose a simulator (e.g., iPhone 15) and press Run (⌘R).

```

import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var billField: UITextField!
    @IBOutlet weak var tipSlider: UISlider!
    @IBOutlet weak var tipLabel: UILabel!
    @IBOutlet weak var totalLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        tipLabel.text = "Tip: 15%"
        tipSlider.value = 15
    }
    @IBAction func sliderChanged(_ sender: UISlider) {
        tipLabel.text = "Tip: \(Int(sender.value))%"
    }
    @IBAction func calculateTapped(_ sender: UIButton) {
        guard let bill = Double(billField.text ?? "") else { // validate input
            totalLabel.text = "Enter a valid amount"
            return
        }
        let tip = bill * Double(tipSlider.value) / 100
        totalLabel.text = String(format: "Tip %.2f | Total %.2f", tip, bill + tip)
        view.endEditing(true)
    }
}

```

Listing 8.9: TipCalc – ViewController.swift

SwiftUI Version (for comparison)

```

import SwiftUI
struct ContentView: View {
    @State private var bill = ""
    @State private var tip = 15.0
    var body: some View {
        VStack(spacing: 16) {
            TextField("Bill amount", text: $bill).keyboardType(.decimalPad)
                .textFieldStyle(.roundedBorder)
            Slider(value: $tip, in: 0...30, step: 1)
            Text("Tip: \(Int(tip))%")
            Text("Total: \(Double(bill) ?? 0) * (1 + tip / 100), specifier: "%.2f")")
        }.padding()
    }
}

```

Listing 8.10: The same idea in SwiftUI (declarative UI, no storyboard)

8.11 Running on the Simulator

- Select a device (e.g., iPhone 15) from the **scheme/destination** menu at the top of Xcode.
- Press **Run** ► or **R**. Xcode builds the app, launches the Simulator and installs the app.
- Simulator tools: rotate (⌘←/→), Home (⌘H), take screenshot (⌘S), simulate location, Dark Mode, slow animations, memory warning.
- Use **Debug area / console** for *print()* output, breakpoints to pause execution, and the **View Debugger** to inspect the hierarchy.

- To run on a real iPhone: connect the device, select it as the destination, set a signing Team under *Signing & Capabilities*, and trust the developer profile on the phone (Settings → General → VPN & Device Management).
- Simulator limits: no real camera, limited sensors, different performance from real hardware; always test on a physical device before release.

Example — Common errors

The app shows a black screen or crashes at launch. What should you check?

Solution

(1) Initial View Controller arrow is missing or Custom Class not set to *ViewController* in the storyboard. (2) A broken outlet connection. (3) Force-unwrapping a nil optional. (4) Signing team not selected for a real device.

Key Takeaways

- ✓ iOS/iPadOS are Apple's mobile systems with four layers: Cocoa Touch, Media, Core Services, Core OS; apps are built in Xcode (macOS) with Swift.
- ✓ Swift basics: let/var, optionals, closures, structs vs classes, protocols.
- ✓ UIKit apps use MVC; views form a hierarchy positioned with Auto Layout constraints.
- ✓ Storyboards define scenes, view controllers control them, segues navigate between them.
- ✓ Connect UI to code with @IBOutlet (reference) and @IBAction (event); run and debug on the Simulator or a real device.

— End of Unit 8 —