CSIT242 Mobile Application Development

LECTURE 1 - INTRODUCTION

Outline

- Introduction
  - What you need to know about lectures, labs, assignments and exams
  - Android Overview
  - Android Studio
  - iOS Overview
  - Xcode
- Mobile Application Development
  - Design Patterns
  - Mobile Application Lifecycle
  - App's Components/Objects

Contact and Subject Organisation

- Lecturer: Dr. Nan Li
- Contact email: nanl@uow.edu.au

- Discussion forums will be opened at Moodle.
- Email & Zoom
  - My best contact: Email or Zoom is preferred if the forums do not suit you.

- Lectures: 7 lectures
- Laboratories: 8 labs
  - There are 5 marking lab exercises

- Assignments: 2 assignments

Assessment

Assessment Items	Percentage of Final Mark		Due Date
	Marks for the Item	Minimum required for a pass	
Lab Exercises 1-5	6 each	N/A	As scheduled
Assignment 1 & 2	10+10 = 20		As scheduled
Final Exam	50	20	Exam week as per schedule
Total	**100**	**50**	**The mark must be ≥50 to pass the subject**

Assessment Due

Assessment Items	Due Date
Lab Exercises 1	15 July 2024 + lab demonstration
Lab Exercises 2	21 July 2024 + lab demonstration
Lab Exercises 3	1 August 2024 + lab demonstration
Lab Exercises 4	4 August 2024 + lab demonstration
Lab Exercises 5	12 August 2024 + lab demonstration
Assignment 1	12 August 2024
Assignment 2	26 August 2024
Final Exam	Exam week as per schedule

Lectures

- The lectures will introduce fundamental concepts and the principles of mobile application design, development and implementation.
- The lectures will contain a sufficient number of examples to facilitate explanation of complex technical aspects.
- It is highly recommended that you implement all examples, compile and run the programs on your computer.
- You are encouraged to participate actively in the lecture sessions answering questions and making your own notes that will help you to better understand the material.

Labs

- The laboratory sessions will be after the lectures.
- All lab tasks will be on the subject Moodle site.
- Students are expected to complete the lab tasks during a lab session. If more time is required to complete all exercises, this can be done at home.
- During the labs, the tutor will assess your solution (for the given lab task) and give you a mark according to the quality of your solution and the level of your understanding.

Additional materials

- Additional materials (readings and videos) will be posted on the subject website.

Assignments

- There will be two programming assignments.
- When an assignment is released, download the assignment description from the subject web site. Read carefully the specifications. Make sure you understand the requirements.
- You may work at home or in the laboratory (when it's available).
- Your solutions must be submitted electronically via the subject website (Moodle). No submission via email will be accepted.
- Late assignments will not be accepted without a granted special consideration.
- Exact time after which the submitted assignment will not be accepted will be indicated in every assignment.

Assignments

- All assignments must be completed individually
- When you submit an assessment task, you are declaring the following:
  - It is your own work and you have not copied anything from others and you have not discussed your work with others.
  - You have not plagiarised from published work (including various internet sources).
  - You have read your responsibilities under the UOW's policy on plagiarism and you understand possible consequences.
  - You have not used storage devices which can be accessed by others without passwords.
- Plagiarism = Big problems
- You may be asked to have a formal meeting with the lecturer to explain your assignment solution if there are doubts that you worked on your assignment yourself

Subject Website

- All important notices related to CSIT242 will be posted on the Subject web site.
- Check it frequently!
- **Note: Any information posted to the subject website is deemed to have been notified to all students!**
- The discussion forum on the subject website can be used by students to discuss only subject-related issues!
- The forum is not to be used to answer any assignment questions. Students who publish their assignment solutions, or provide assistance to other students may get 0 for their assignment.
- Forum posts that are not related to the subject will be deleted!

Self-directed Study

- **Listening passively is useless!**
- Download lecture notes from the subject website and look through the lecture notes prior to lectures.
- Attend all lectures. Take your own notes and add your own comments or questions during the lectures.
- Read/watch/implement all materials/solutions posted on the subject website.
- Implement examples discussed at lectures or developed during labs on your computer at home.
- If you have any questions, attend **consultations**.

Subject Materials

- Recommended books:
  - SwiftUI Essentials - iOS 16 Edition: Learn to Develop iOS Apps Using SwiftUI, Swift, and Xcode 14, Neil Smyth, Payload Media Inc., 2022.
  - iOS 15 Programming Fundamentals with Swift - Swift, XCode and Cocoa Basics, Matt Neuburg, O'Reilly Media Inc., 2021.
  - iOS 17 Programming for Beginners - 8th Edition, Ahmad Sahar, Packt, 2023.
  - Android Studio Electric Eel Essentials - Java Edition, Neil Smyth, Payload Media Inc., 2023.
  - Android Studio 4.1 Essentials - Java Edition, Neil Smyth, Payload Media Inc., 2020.
  - Learn Android Studio 4: Efficient Java-Based Android Apps Development, Ted Hagos, Apress, 2nd ed., 2020.
  - Programming iOS 14: Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2020.
  - Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017.
- Lecture notes & Labs exercises:
  - The lecture notes are available on the subject website (The lecture notes may not include some examples and explanations given in lectures).
  - The lab exercises are available on the subject website.
- Additional materials
  - Additional materials may be posted on the subject website.

Software Requirements

- Xcode v15.2 (Mac)
- Android Studio (Electrical Eel 2022.1.1) (Mac and Windows) including
  - Android Emulator
  - Android SDK Build Tools (33.0.1),
  - Android SDK Tools (33),
  - Android SDK Platform-Tools,
  - Gradle (7.5),
  - Android Gradle plugin,
  - Google Play services libraries (Google Play services added in Android SDK)
  - HAXM installer (may be needed for the Android Emulator)

Android - History & Overview

- Developed by Android Inc.
- Bought by Google in 2005
- Majority of Android software is licensed with Apache 2.0
- Version numbers (before version 10) are associated alphabetically with desserts
- Developers program using an API level associated with a specific version (shown in next slide)

Android - History & Overview

Version	Codename	API level
1.0		1
1.1		2
1.5	Cupcake	3
1.6	Donut	4
2.0	Eclair	5
2.01	Eclair	6
2.1	Eclair	7
2.2.x	Froyo	8
2.3 - 2.3.2	Gingerbread	9

Hardly anyone uses these versions.

...

...

...

Check Android versions preview at:
https://developer.android.com/about/dashboards

Android - History & Overview

Current status on Platform and Versions

The most recent stable version of Android is Android 14, which was released in October 2023 (API 34).

source: https://en.wikipedia.org/wiki/Android_version_history
https://apilevels.com/

Name	Internal codename[11]	Version number(s)	API level	Initial stable release date	Latest security patch date[16]	Latest Google Play Services version[17][18] (release date)
Android 1.0	—	1.0	1	September 23, 2008	—	—
Android 1.1	Petit Four	1.1	2	February 9, 2009		
Android Cupcake	Cupcake	1.5	3	April 27, 2009		
Android Donut	Donut	1.6	4	September 15, 2009		
Android Eclair	Eclair	2.0	5	October 27, 2009		
		2.0.1	6	December 3, 2009		
		2.1	7	January 11, 2010[19]		
Android Froyo	Froyo	2.2 – 2.2.3	8	May 20, 2010		3.2.25 (October 2014)
Android Gingerbread	Gingerbread	2.3 – 2.3.2	9	December 6, 2010		10.0.84 (November 2016)
		2.3.3 – 2.3.7	10	February 9, 2011		
Android Honeycomb	Honeycomb	3.0	11	February 22, 2011		
		3.1	12	May 10, 2011		
		3.2 – 3.2.6	13	July 15, 2011		
Android Ice Cream Sandwich	Ice Cream Sandwich	4.0 – 4.0.2	14	October 18, 2011		14.8.49 (February 2019)
		4.0.3 – 4.0.4	15	December 16, 2011		
Android Jelly Bean	Jelly Bean	4.1 – 4.1.2	16	July 9, 2012		21.33.56 (September 2021)
		4.2 – 4.2.2	17	November 13, 2012		
		4.3 – 4.3.1	18	July 24, 2013		
Android KitKat	Kitkat	4.4 – 4.4.4	19	October 31, 2013	October 2017	23.30.13 (August 2023)
		4.4W – 4.4W.2	20	June 25, 2014	?	
Android Lollipop	Lolipop	5.0 – 5.0.2	21	November 4, 2014[20]	November 2017	23.43.13 (November 2023)
		5.1 – 5.1.1	22	March 2, 2015[21]	March 2018	
Android Marshmallow	Marshmallow	6.0 – 6.0.1	23	October 2, 2015[22]	August 2018	
Android Nougat	Nougat	7.0	24	August 22, 2016	August 2019	
		7.1 – 7.1.2	25	October 4, 2016	October 2019	
Android Oreo	Oreo	8.0	26	August 21, 2017	January 2021	
		8.1	27	December 5, 2017	October 2021	
Android Pie	Pie[23]	9	28	August 6, 2018	January 2022	
Android Q	Queen Cake[24]	10	29	September 3, 2019	February 2023	
Android 11	Red Velvet Cake[24]	11	30	September 8, 2020	January 2024	
Android 12	Snow Cone	12	31	October 4, 2021		
Android 12L	Snow Cone v2	12.1[a]	32	March 7, 2022		
Android 13	Tiramisu	13	33	August 15, 2022		
Android 14	Upside Down Cake[27]	14	34	October 4, 2023		
Android 15	Vanilla Ice Cream[28]	15	[to be determined]			Q3 2024 — —

Legend: Old version | Older version, still maintained | Latest version | Latest preview version | Future release

Version	SDK / API level	Version code	Codename	Cumulative usage[1]	Year[4]
Android 15 DEV	Level 35	*TBD*	Vanilla Ice Cream[2]	—	*TBD*
Android 14	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake[2]	9.3%	2023
	▪ `targetSdk` will need to be 34+ for new apps and app updates by August 31, 2024.				
Android 13	Level 33	TIRAMISU	Tiramisu[2]	41.6%	2022
	▪ `targetSdk` must be 33+ for new apps and app updates since August 31, 2023.				
Android 12	Level 32 Android 12L	S_V2	Snow Cone[2]	58.9%	
	Level 31 Android 12	S			2021
Android 11	Level 30	R	Red Velvet Cake[2]	75.4%	2020
Android 10	Level 29	Q	Quince Tart[2]	84.3%	2019
Android 9	Level 28	P	Pie	90.3%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	92.4%	2017
	Level 26 Android 8.0	O		95.2%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	95.6%	2016
	Level 24 Android 7.0	N		97.1%	
Android 6	Level 23	M	Marshmallow	98.5%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	99.3%	
	Level 21 Android 5.0	LOLLIPOP, L		99.5%	2014
	▪ Jetpack Compose requires a `minSdk` of 21 or higher. ▪ Google Play services require API level 21 and up since August 2023.				
Android 4	Level 20 Android 4.4W[3]	KITKAT_WATCH	KitKat	99.7%	
	Level 19 Android 4.4	KITKAT			2013
	▪ Jetpack/AndroidX libraries require a `minSdk` of 19 or higher since October 2023.				
	Level 18 Android 4.3	JELLY_BEAN_MR2	Jelly Bean	99.8%	
	Level 17 Android 4.2	JELLY_BEAN_MR1		99.8%	2012
	Level 16 Android 4.1	JELLY_BEAN		99.8%	
	Level 15 Android 4.0.3 – 4.0.4	ICE_CREAM_SANDWICH_MR1	Ice Cream Sandwich	99.8%	2011
	Level 14 Android 4.0.1 – 4.0.2	ICE_CREAM_SANDWICH			
Android 3	Level 13 Android 3.2	HONEYCOMB_MR2	Honeycomb	*No data*	
	Level 12 Android 3.1	HONEYCOMB_MR1			
	Level 11 Android 3.0	HONEYCOMB			
Android 2	Level 10 Android 2.3.3 – 2.3.7	GINGERBREAD_MR1	Gingerbread		
	Level 9 Android 2.3.0 – 2.3.2	GINGERBREAD			2010
	Level 8 Android 2.2	FROYO	Froyo		
	Level 7 Android 2.1	ECLAIR_MR1	Eclair		
	Level 6 Android 2.0.1	ECLAIR_0_1			2009
	Level 5 Android 2.0	ECLAIR			
Android 1	Level 4 Android 1.6	DONUT	Donut		
	Level 3 Android 1.5	CUPCAKE	Cupcake		
	Level 2 Android 1.1	BASE_1_1	Petit Four		
	Level 1 Android 1.0	BASE	*None*		2008

Android - History & Overview

- Android is a comprehensive open source platform based on Linux and championed by Google.
- It's a powerful development framework that includes everything you need to build great apps using a mix of
  - Java/Kotlin and XML.
- Android provides an adaptive app framework that allows you to provide unique resources for different device configurations.
- Android does not use a Java Virtual Machine. Android does not use Abstract Window Toolkit or Swing library.
- Android used Dalvik, specialized VM, now is using ART – Android Runtime.

Android - Overview

- Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java or Kotlin language environment.
- Apps provide multiple entry points.
- Android apps are built as a combination of distinct components that can be invoked individually.
  - For instance, an individual activity provides a single screen for a user interface, and a service independently performs work in the background.
- Android provides an adaptive app framework that allows you to provide unique resources for different device configurations. Apps adapt to different devices.
  - For example, you can create different XML layout files for different screen sizes and the system determines which layout to apply based on the current device's screen size.

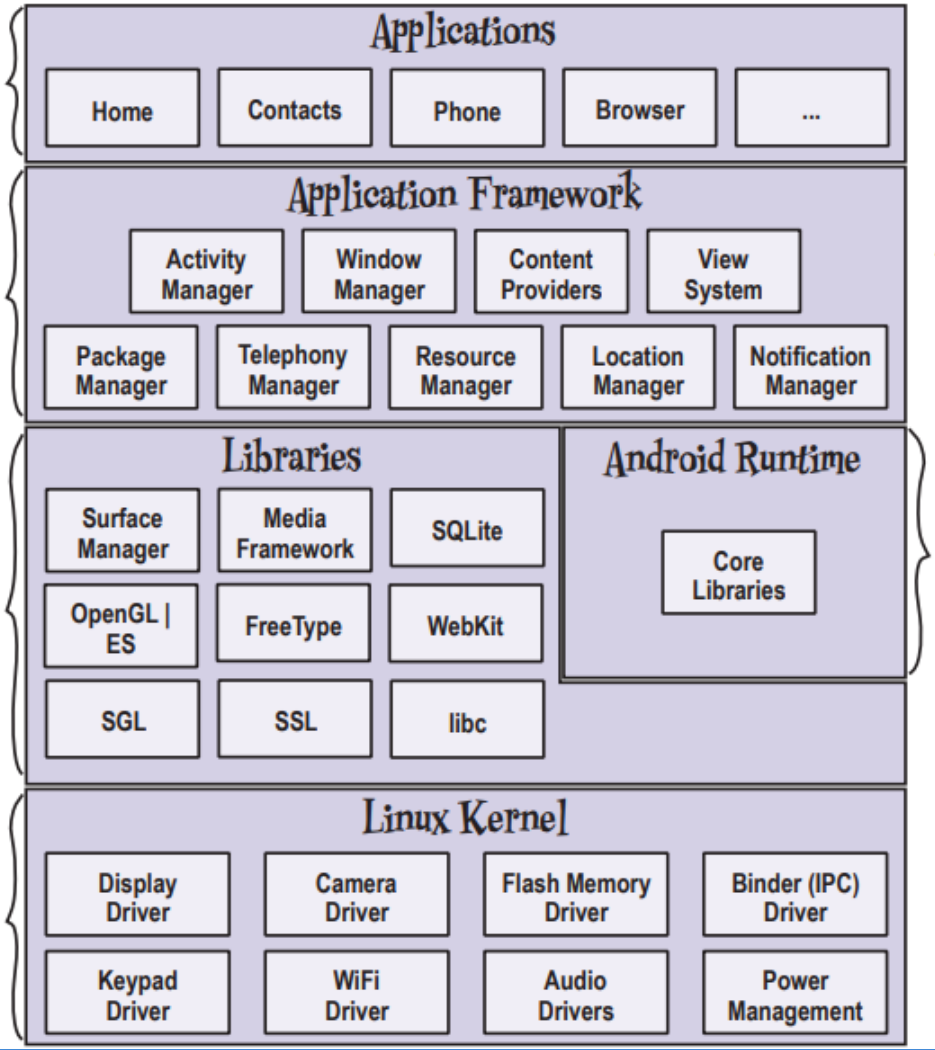
Android - architecture

Android comes with a set of core applications such as Contacts, Phone, Calendar, and a browser.

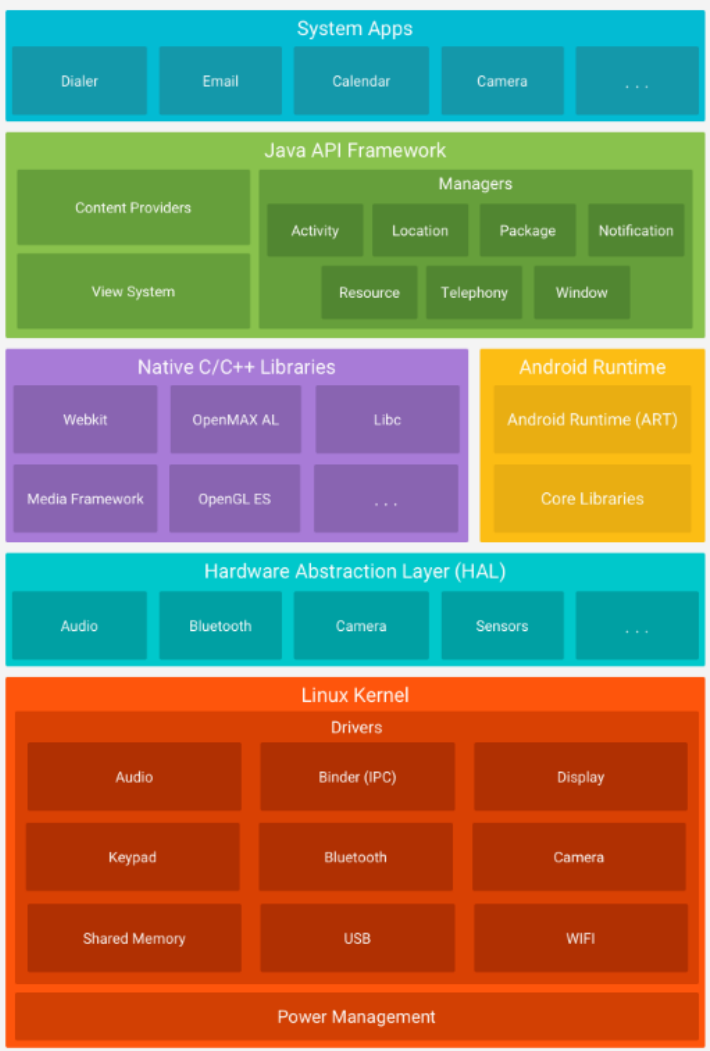
When you build apps, you have access to the same APIs used by the core applications. You use these APIs to control what your app looks like and how it behaves.

Underneath the application framework lies a set of C and C++ libraries. These libraries get exposed to you through the framework APIs.

Underneath everything else lies the Linux kernel. Android relies on the kernel for drivers, and also core services such as security and memory management.



The Android runtime comes with a set of core libraries that implement most of the Java programming language. Each Android app runs in its own process.

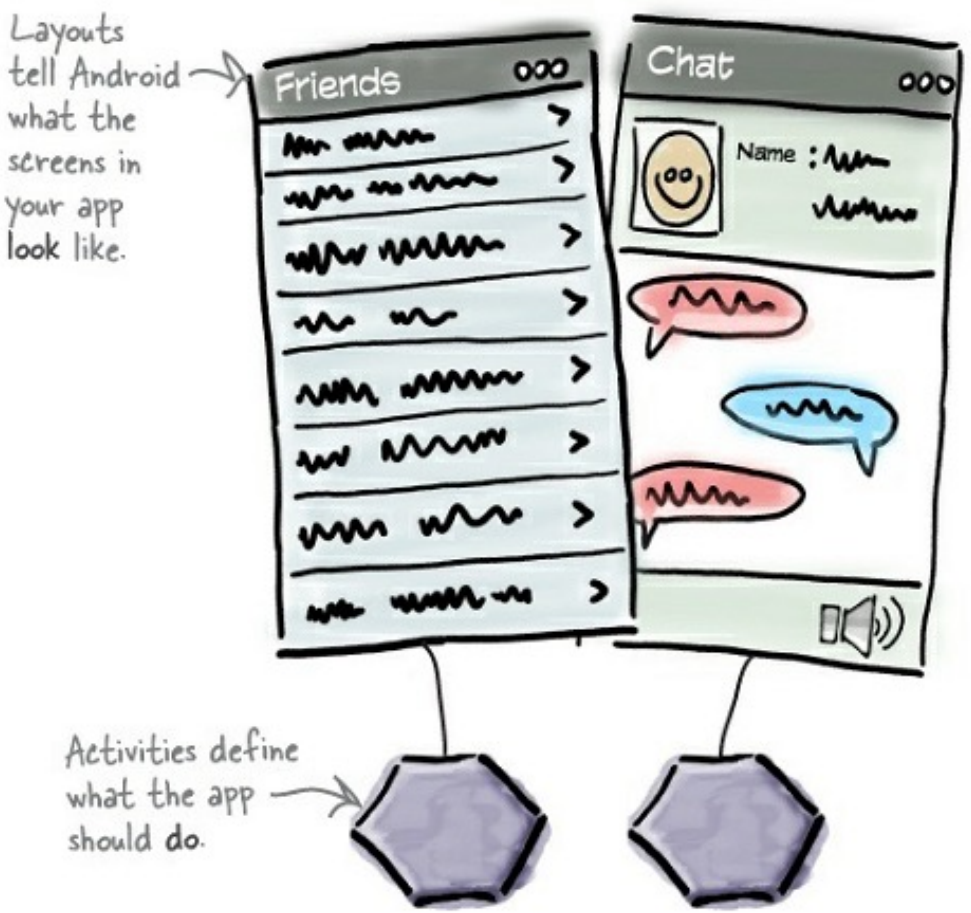


Source: Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017.

https://developer.android.com/guide/platform

Typical Android Applications

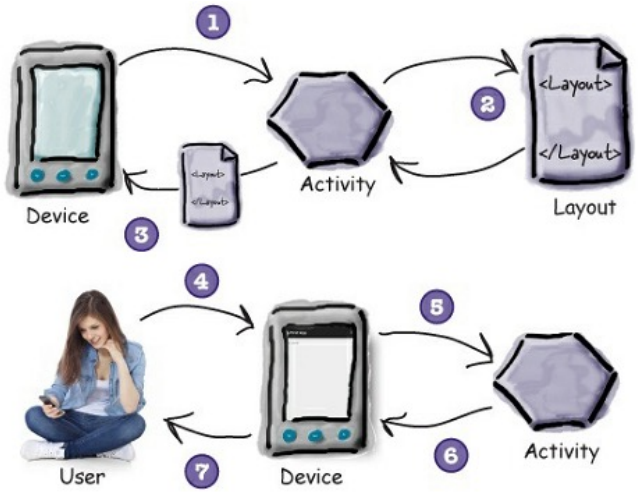
- A typical Android app is comprised of one or more screens. You define what each screen looks like using a layout to define its appearance.
- Layouts are usually defined using XML, and can include GUI components such as buttons, text fields, and labels.



Source: Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017.

Android Applications Behavior

- Every Android app is a collection of screens, and each screen is comprised of an activity and a layout.
  - The device launches your app and creates an activity object.
  - The activity object specifies a layout.
  - The activity tells Android to display the layout on screen.
  - The user interacts with the layout that's displayed on the device.
  - The activity responds to these interactions by running application code.
  - The activity updates the display...
  - ...which the user sees on the device.



Source: Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017.

Android Development Tools

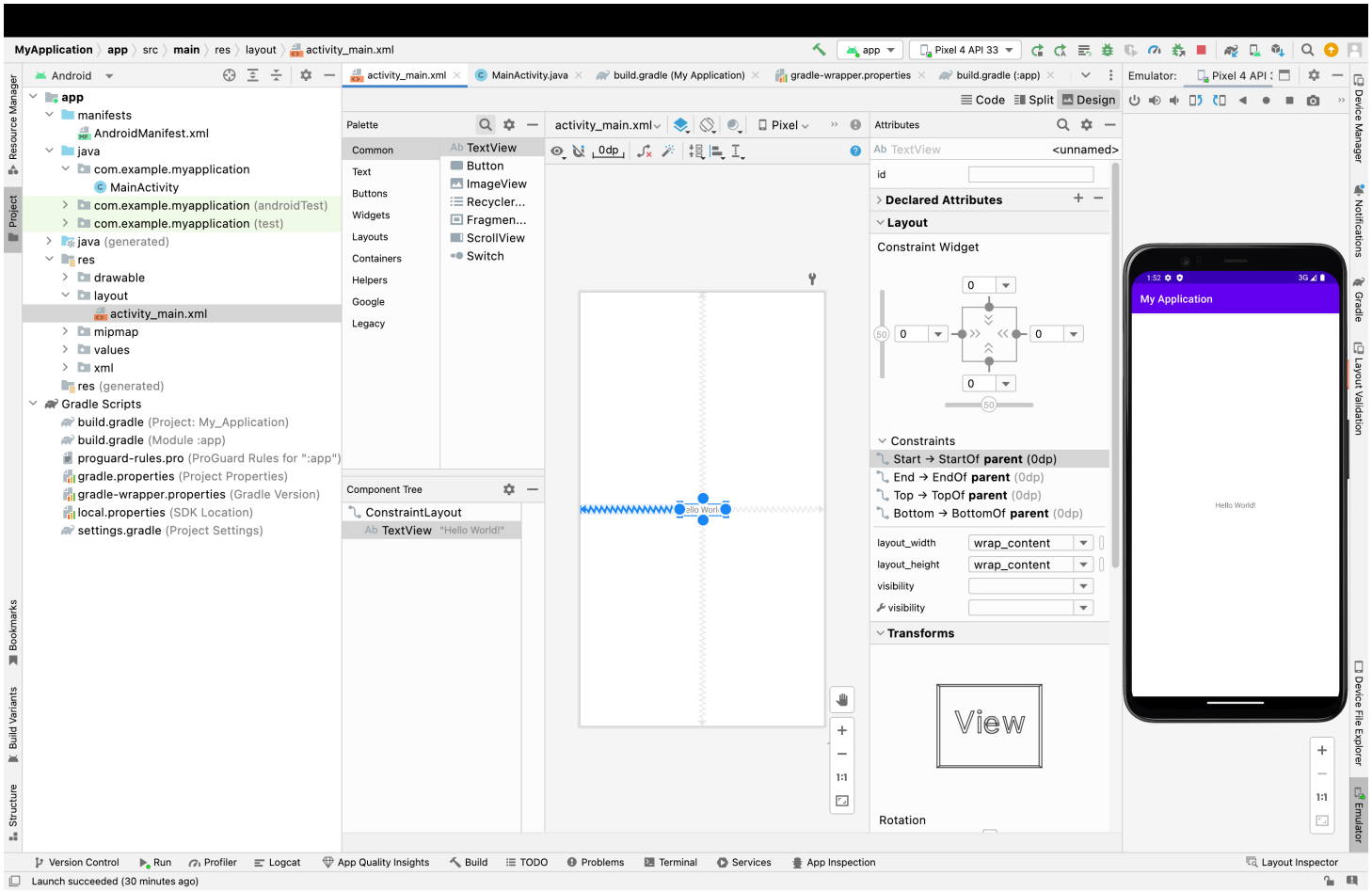
- **Android Studio** - is the official IDE for Android application development, based on IntelliJ IDEA.

Android Studio

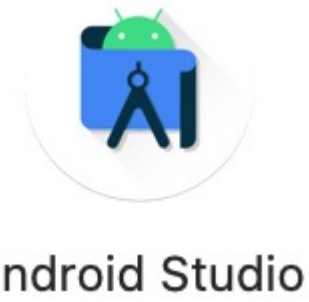
- Android Studio offers:
  - Flexible Gradle-based build system;
  - Build variants and multiple apk file generation;
  - Code templates to help you build common app features;
  - Rich layout editor with support for drag and drop theme editing;
  - Tools to catch performance, usability, version compatibility, and other problems;
  - ProGuard and app-signing capabilities;
  - Built-in support for Google Cloud Platform…
- For the code files we will use **Java**.

Android Studio

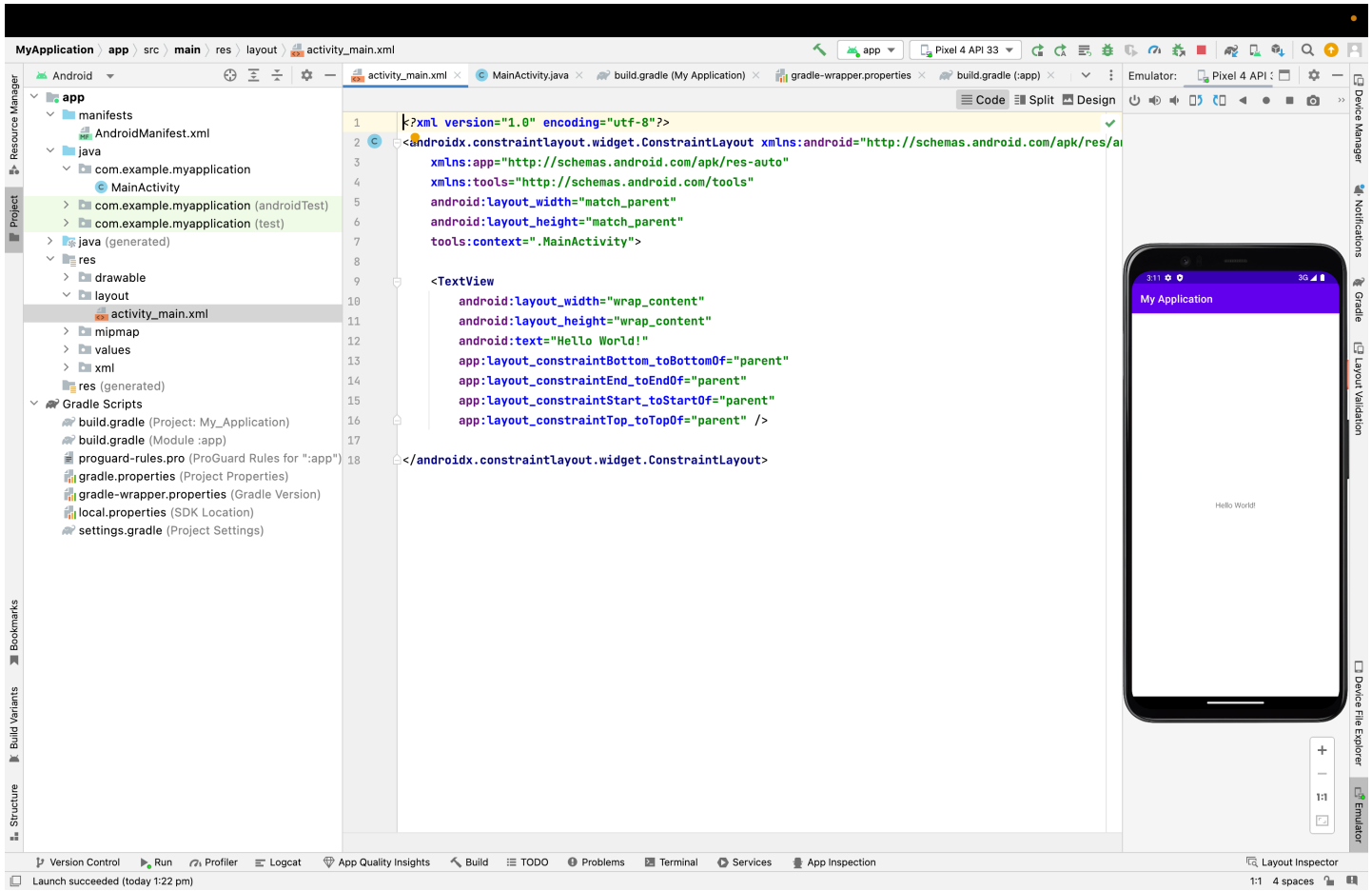
- Design View



Android Studio



- Text View (Code)



iOS - History & Overview

- iPhone released in 2007
- In June 2010 Apple rebranded iPhone OS as iOS
- Software Development Kit released in 2008
- Developers make iOS apps using the iOS SDK and Xcode
- Applications can be developed in Objective-C and Swift
- Each major version of iOS comes with the next version of Xcode
- Current version is iOS 17 and Xcode 15
- All development resources come from Apple (https://developer.apple.com/)

iOS - History & Overview

V·T·E

Overview of iOS and iPadOS versions

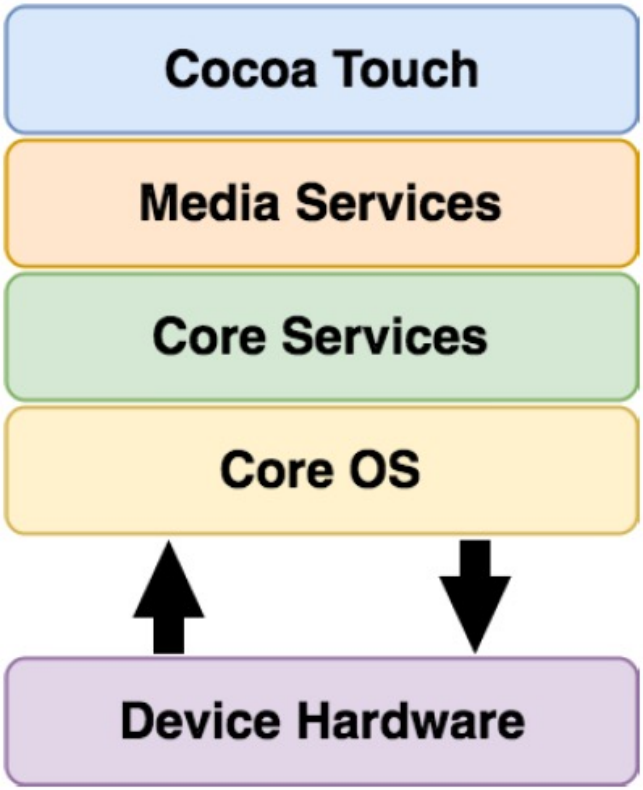
Version	Initial release date	Latest version	Release date	Device end-of-life		
				iPad	iPhone	iPod Touch
iPhone OS 1	June 29, 2007	1.1.5	July 15, 2008	—	—	—
iPhone OS 2	July 11, 2008	2.2.1	January 27, 2009	—	—	—
iPhone OS 3	June 17, 2009	3.2.2	February 2, 2010	—	1st[a]	1st[a]
iOS 4	June 21, 2010	4.3.5[b]	July 25, 2011	—	3G[c]	2nd[c]
iOS 5	October 12, 2011	5.1.1	May 7, 2012	1st	—	3rd
iOS 6	September 19, 2012	6.1.6	February 21, 2014	—	3GS	4th
iOS 7	September 18, 2013	7.1.2	June 30, 2014	—	4	—
iOS 8	September 17, 2014	8.4.1	August 13, 2015	—	—	—
iOS 9	September 16, 2015	9.3.6	July 22, 2019	2, 3rd, Mini[d]	4S	5th[d]
iOS 10	September 13, 2016	10.3.4	July 22, 2019	4th[e]	5, 5C[e]	—
iOS 11	September 19, 2017	11.4.1	July 9, 2018	—	—	—
iOS 12	September 17, 2018	12.5.7	January 23, 2023	Air (1st), Mini 2, Mini 3	5S, 6	6th
iOS 13 / iPadOS 13	September 19, 2019 (iOS) September 24, 2019 (iPadOS)	13.7	September 1, 2020	—	—	—
iOS 14 / iPadOS 14	September 16, 2020	14.8.1	October 26, 2021	—	—	—
iOS 15 / iPadOS 15	September 20, 2021	15.8.1	January 22, 2024	Air 2, Mini 4	6S, SE (1st), 7	7th
iOS 16 / iPadOS 16	September 12, 2022 (iOS) October 24, 2022 (iPadOS)	16.7.5	January 22, 2024	Pro (1st), 5th	8, X	—
iOS 17 / iPadOS 17	September 18, 2023	17.3.1	February 8, 2024	—	—	—

Legend: Obsolete Supported Current

Source: https://en.wikipedia.org/wiki/IOS_version_history

iOS - Architecture

- Architecture of iOS is a layered architecture.
- At the uppermost level iOS works as an intermediary between the underlying hardware and the apps you make.
- Apps do not communicate to the underlying hardware directly.
- Apps talk with the hardware through a collection of well defined system interfaces. These interfaces make it simple to write apps that work constantly on devices having various hardware abilities.
- Lower layers gives the basic services which all application relies on and higher level layer gives sophisticated graphics and interface related services.



Source: iOS 12 App Development Essentials, Neil Smyth, Payload Media, Inc., 2018.

iOS - Architecture

- The Cocoa touch layer incorporates frameworks for building iOS applications.
- Media layer offer technologies that incorporates graphics, audio and video.
- Core Services layer technologies provide essential services to apps but have no direct bearing on the app's user interface.
- The Core OS layer holds the low-level features that most other technologies are built upon.

iOS - Architecture

Cocoa Touch Layer	EventKit framework; GameKit Framework; iAd Framework; MapKit Framework; PushKitFramework; UIKit Framework (Multitasking support, Basic app management and infrastructure, User interface management, Support for Touch and Motion event, Cut, copy and paste support, …) + more
Media Layer	UIKit Graphics; Core Graphics framework; Core Animation; Core Images; OpenGl ES and GLKit; Metal; Media Player Framework; AV Foundation; OpenAL; AV Kit framework; AV Foundation; Core Media, …
Core Services Layer	Address book framework; Cloud Kit framework; Core data Framework; Core Foundation framework; Core Location framework; Core Motion Framework; Foundation Framework; Healthkit framework; Homekit framework; Social framework; StoreKit framework, …
Core OS Layer	Accelerate Framework; Core Bluetooth Framework; External Accessory Framework; Security Services Framework; Local Authentication Framework, …

iOS - Applications

- Every iOS app is a collection of Views and View Controllers.
- Apps need to work with the iOS to ensure that they deliver a great user experience.
- iOS apps should:
  - have good design - app's design and user interface,
  - be fast and responsive,
  - use as little power as possible,
  - need to support all of the latest iOS devices (while still appearing as if the app was tailored for the current device).

iOS Development Tools

- **Xcode** – free from developer.apple.com

- Interface Builder (Built into Xcode 15)
- iOS simulator – installed together with Xcode
- Instruments Debugging
- Apple documentation –
  - include Swift programming
  - document as free iBook

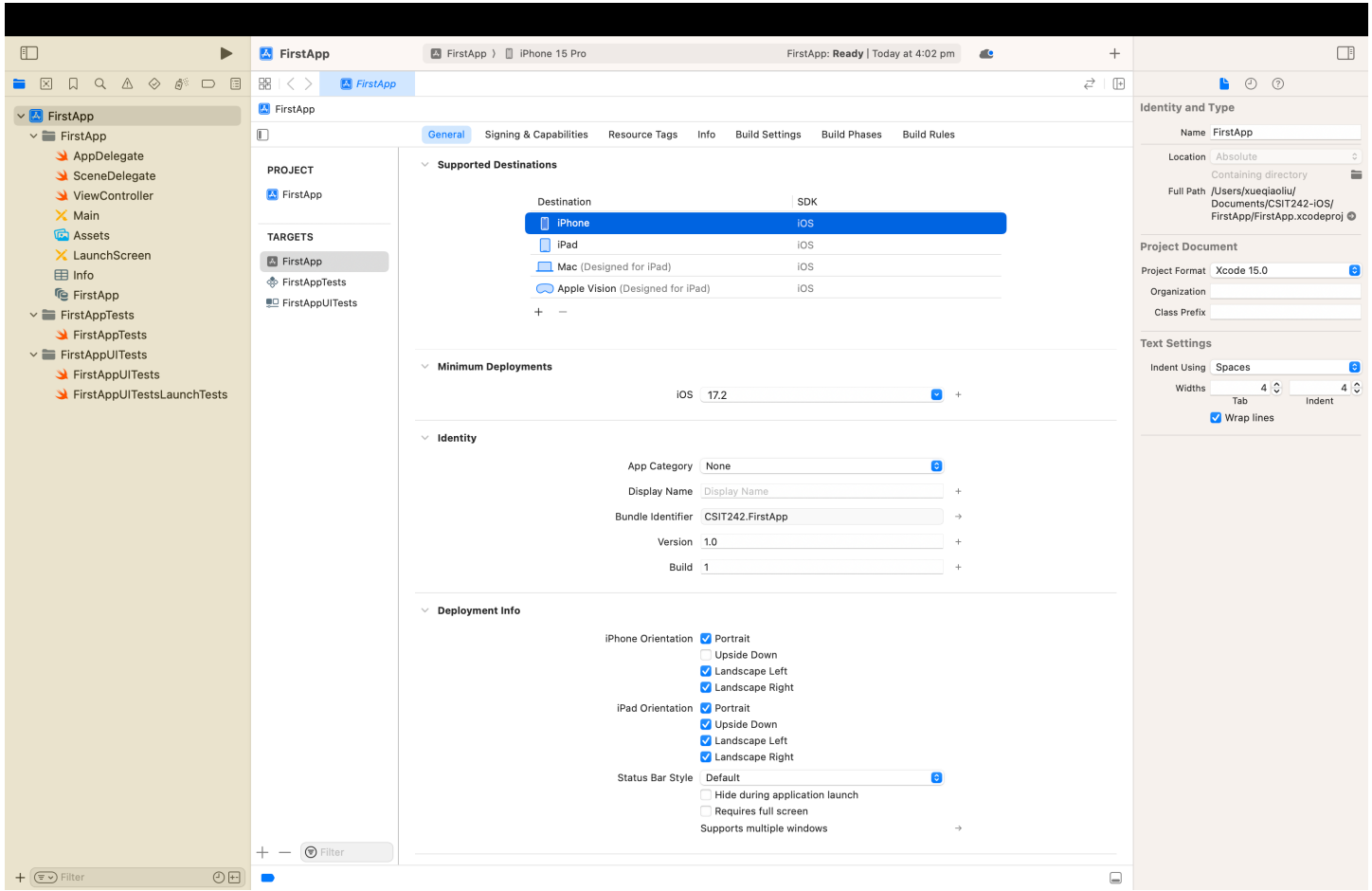
Xcode

Xcode

- Xcode offers:
  - Complete toolset for building OS X and iOS applications,
  - Full featured IDE,
  - Code, Create, Compile, Run,
  - On-the-fly syntax and logic checking Integrated Debugger,
  - Integrated Interface Builder,
  - Drag & Drop User Interface creation Inbuilt visual storyboarding.

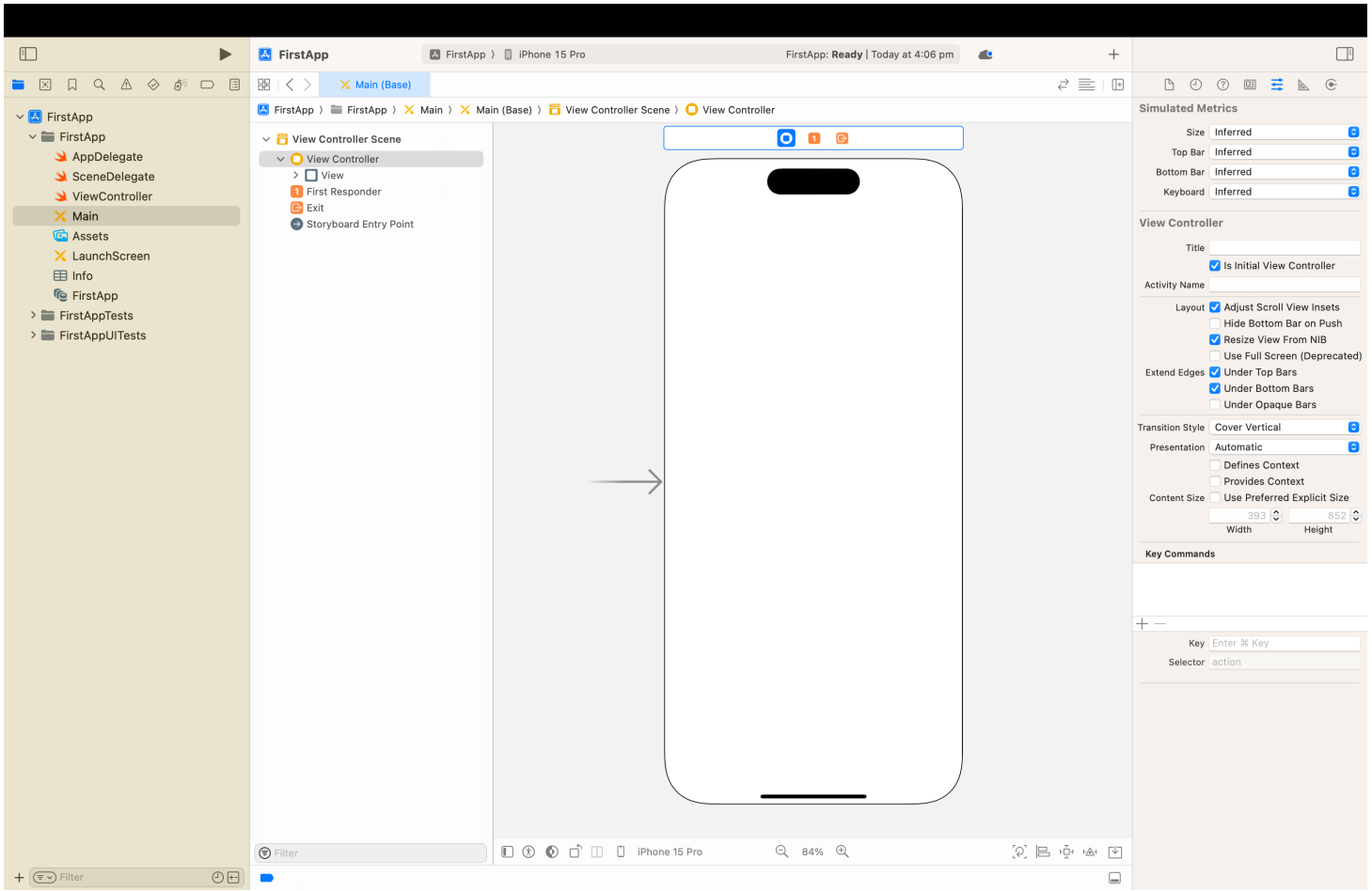
Xcode

- main window



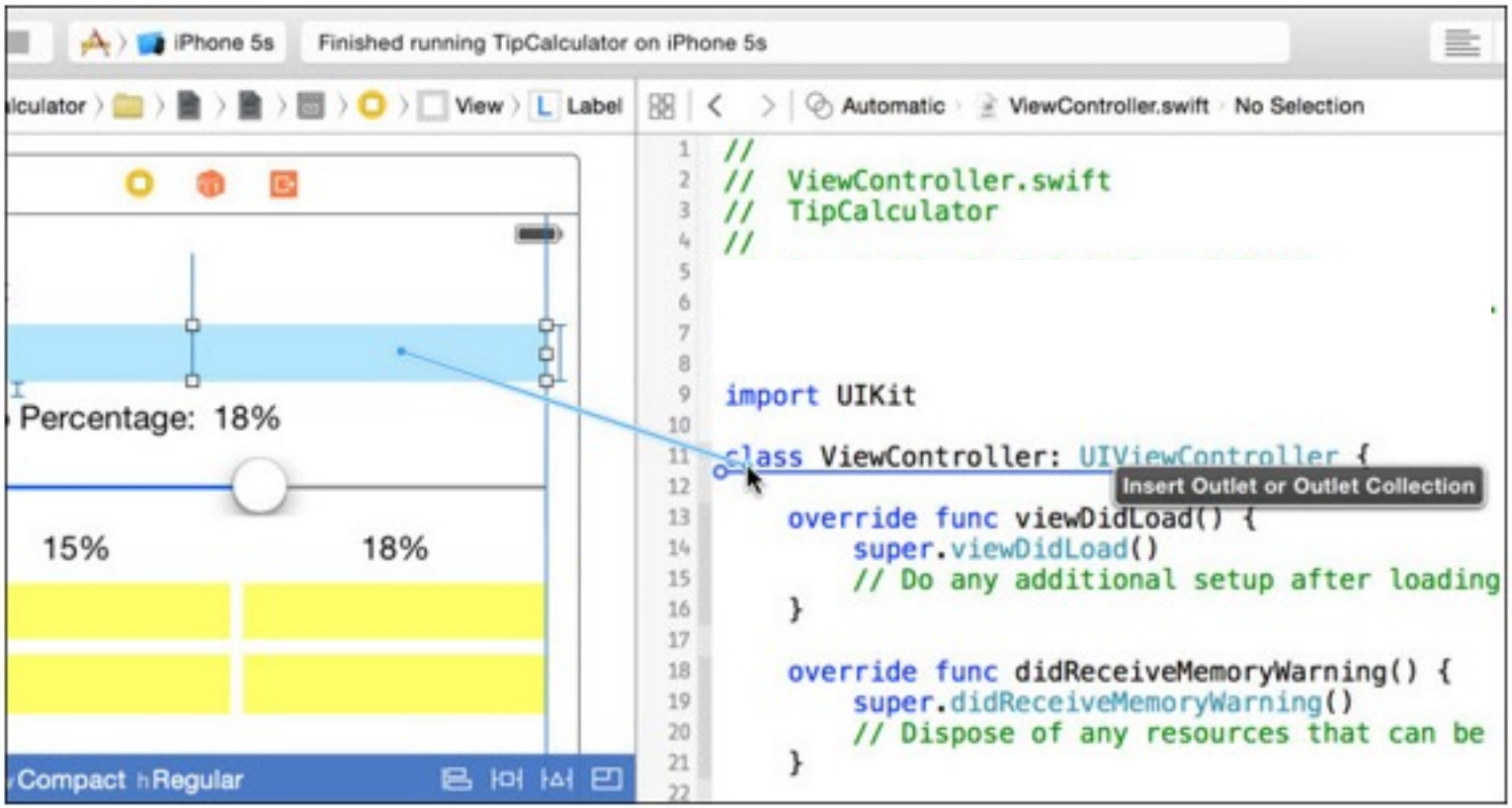
Xcode

- Interface Builder



Xcode

- Assistant Editor



Xcode

- Source Editor
  - Complete with code completion, code folding, syntax highlighting, and message bubbles that display warning, errors, and other context-sensitive information inline with your code.
- Assistant Editor
  - The Assistant button splits the editor in two, creating a secondary pane that automatically displays files that are most helpful to you based on the code you are actively editing. It can show the header counterpart, the superclass, callers, callees, or other helpful files.
- Version Editor
  - Xcode's Version editor displays a running timeline of commits, helps you determine blame, and graphically goes back in time to compare source files, with full support for Subversion and Git source control (SCM) systems.
- Interface Builder Built In
  - Design and test your user interface without writing a line of code, prototype in minutes, then graphically connect your interface to the source within the Xcode editor.
- iOS Simulator
  - With the iOS SDK, Xcode can build, install, run, and debug Cocoa Touch apps in a Mac-based iOS Simulator for a streamlined development workflow.
- Integrated Build System
  - Handles the most complex builds, scaling to maximize the power of multi-core Macs, and will automatically sign, provision, and install iPad and iPhone apps onto a device.

Xcode

- Compilers
  - The powerful open-source compiler for C, C++, Objective-C, Swift is built into Xcode and available from Terminal. With it, your code compiles quickly, and is optimized by Apple.
- Graphical Debugger
  - Debug your app directly within the Xcode editor. Hover over any variable to drill into its contents, use Quick Look to see the data it contains, or right-click to add the variable to the watch list.
- Asset Catalog
  - The asset catalog editor in Xcode manages your app's images, grouping together various resolutions of the same asset. When building, Xcode compiles the asset catalog into the most efficient bundle for final distribution.
- Complete Documentation
  - Search for anything within Xcode and the documentation viewer will find it, either on your Mac or on the Apple Developer website.
- Live Issues
  - Just like a word processor highlights spelling errors, Live Issues highlights common coding mistakes, without the need to click 'build' first.
- Fix-it
  - Xcode goes beyond just reporting errors. When you make a coding mistake, Xcode will immediately alert you, and a single keyboard shortcut will instantly fix the issue, so you won't miss a beat while coding.

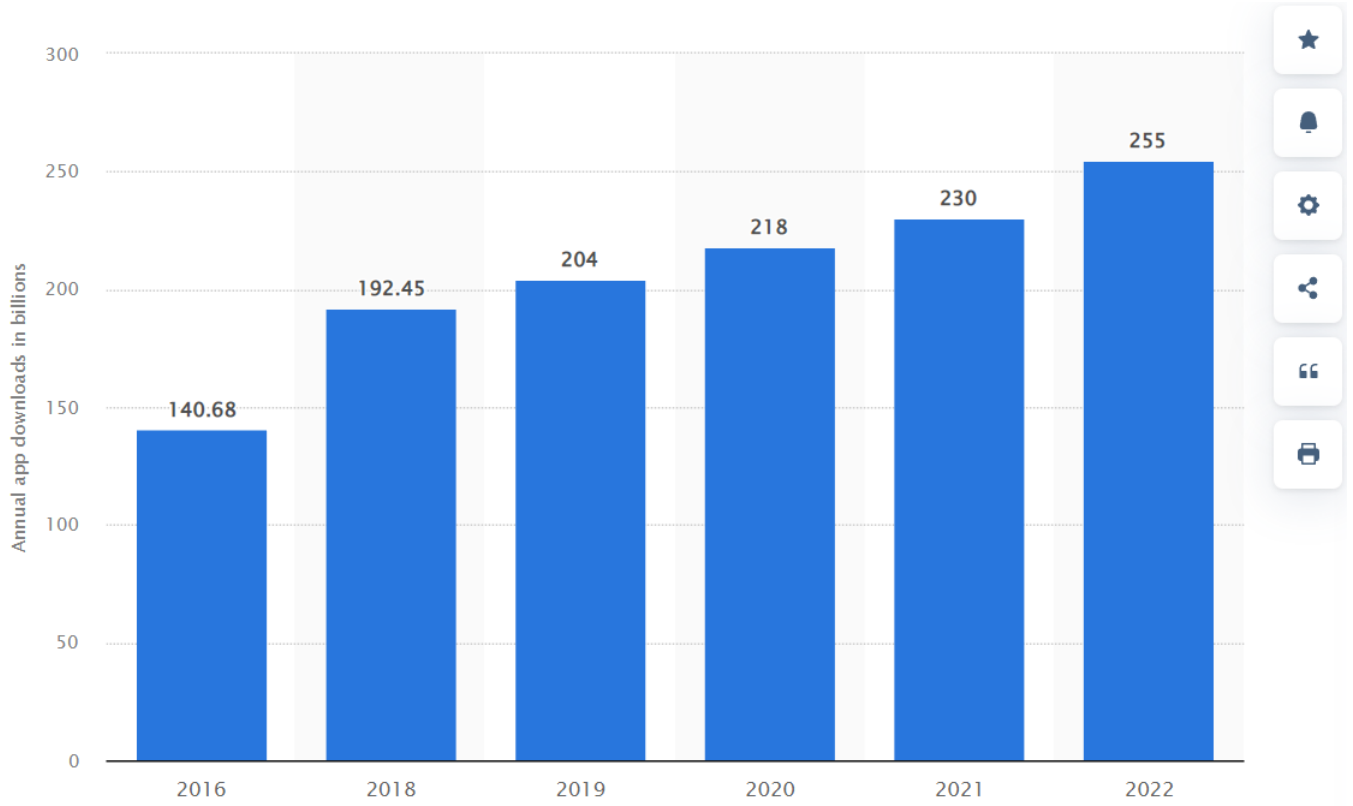
iOS vs Android

iOS	Android
Closed	Open-source
Limited customization	Highly customizable
40 Languages	100+ Languages
macOS	Linux OS family

What about: Internet browsing, Maps, Video chat, Voice commands, Cloud services, Biometric authentication…

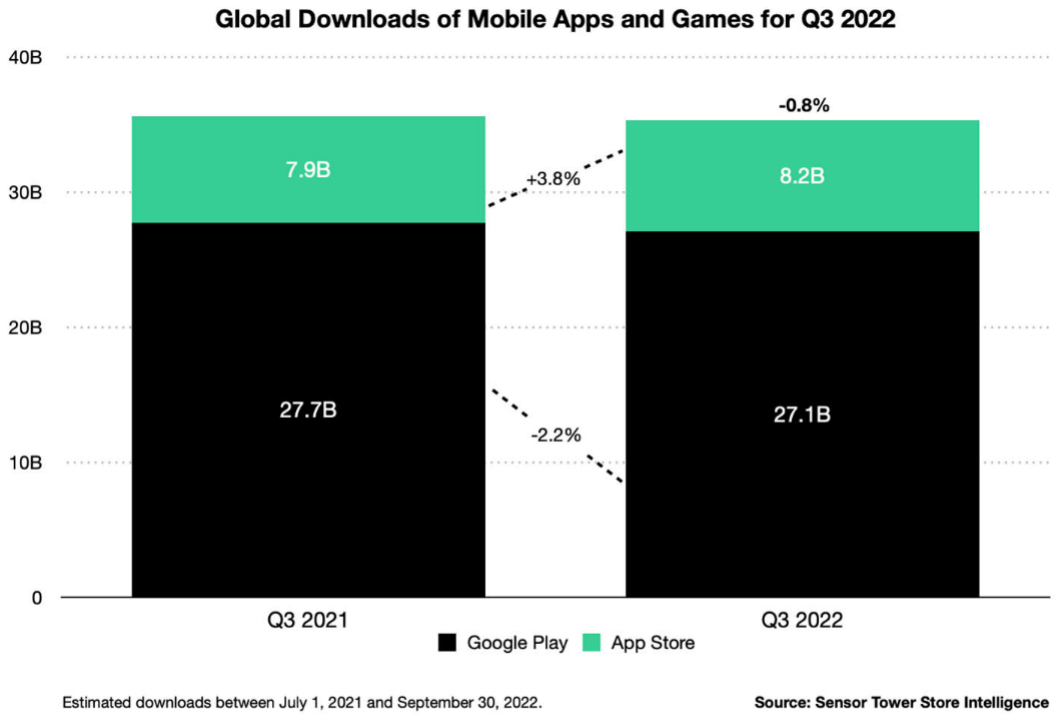
Google Play vs App Store – interesting statistics

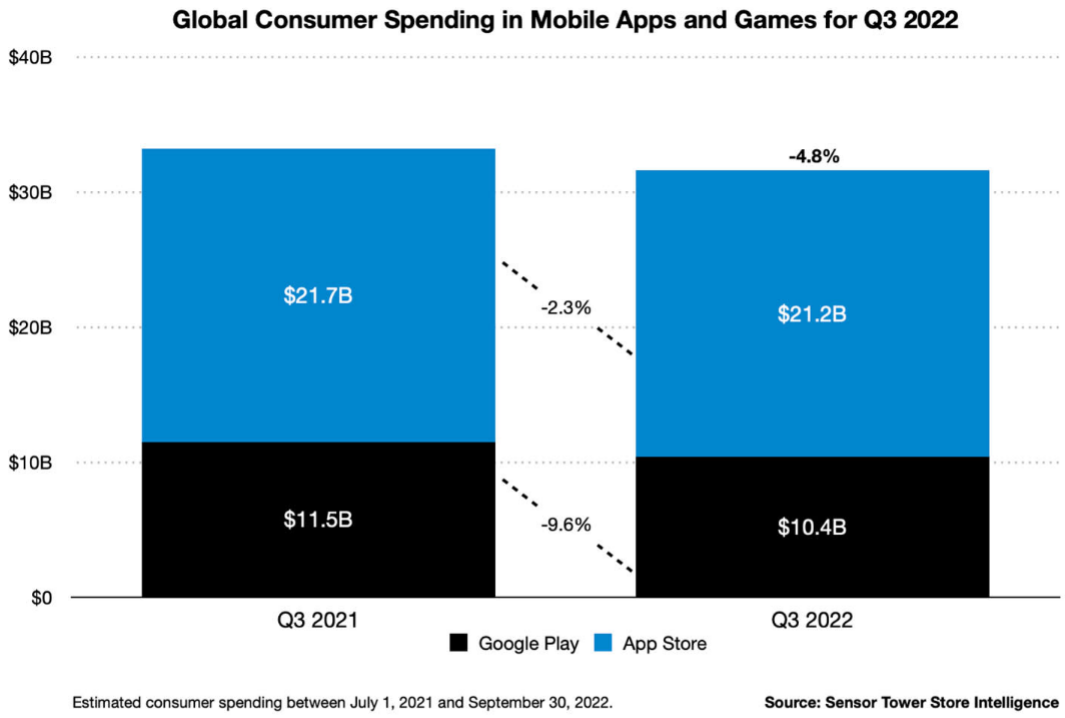
Number of apps available in leading app stores as of 3rd quarter 2022



Source: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

Google Play vs App Store – interesting statistics





Source: https://sensortower.com/blog/app-revenue-and-downloads-q3-2022

New Features in Android 14

- Personalization
- Connectivity
- Productivity
- Entertainment and Media
- Privacy and Security
- Accessibility …

New Features in iOS 17

- StandBy
- Interactive widgets
- Contact Posters
- Live Voicemail
- Stickers
- Journal
- Messages
- Keyboard
- FaceTime
- AirDrop …

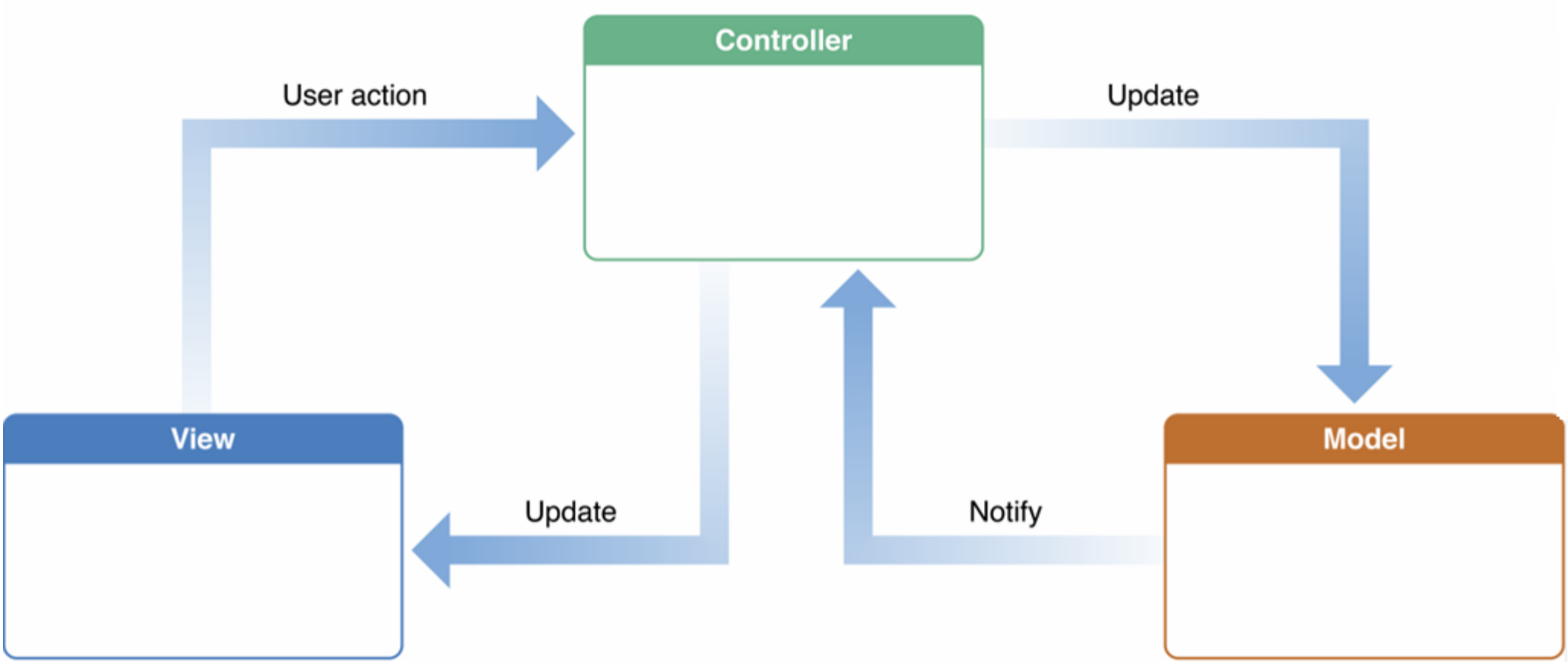
App architecture design

- Good architecture design involves:
  - modularity
  - reusability
  - efficiency
- Concerns
  - testability
  - maintenance
- Challenges
  - limited computing capability and resources (CPU, memory, screen size, battery)

Design patterns: Model-View-Controller (MVC)

- Programming paradigm for iOS and Android application.
- Splits the application into manageable parts and assigns objects in an application in one of three roles:
  - Model
  - View
  - Controller
- Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.
- The collection of objects of a certain MVC type in an application is sometimes referred to as a *layer*.
- MVC is central to a good design for a Cocoa application.
  - Many objects in these applications tend to be more reusable, and their interfaces tend to be better defined.
  - Applications having an MVC design are also more easily extensible than other applications.
  - Moreover, many Cocoa technologies and architectures are based on MVC and require that your custom objects play one of the MVC roles.

Design patterns: Model-View-Controller (MVC)



source: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Design patterns: Model-View-Controller (MVC)

- Model Objects
  - encapsulate the data specific to an application and define the logic and computation that manipulate and process that data (for example, a model object might represent a character in a game or a contact in an address book).
  - A model object can have to-one and to-many relationships with other model objects.
  - Much of the data that is part of the persistent state of the application (whether that persistent state is stored in files or databases) should reside in the model objects after the data is loaded into the application.
  - Model objects represent knowledge and expertise related to a specific problem domain, so they can be reused in similar problem domains.
  - Ideally, a model object should have no explicit connection to the view objects that present its data and allow users to edit that data—it should not be concerned with user-interface and presentation issues.
- User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes, it notifies a controller object, which updates the appropriate view objects.

Design patterns: Model-View-Controller (MVC)

- View Objects
  - is an object in an application that users can see.
  - A view object knows how to draw itself and can respond to user actions.
  - A major purpose of view objects is to display data from the application's model objects and to enable the editing of that data. Despite this, view objects are typically decoupled from model objects in an MVC application.
  - Because we reuse and reconfigure them, view objects provide consistency between applications.
  - Both the UIKit and Android UI frameworks provide collections of view classes, and their respective Interface Builders offers dozens of view objects in its Library.
- View objects learn about changes in model data through the application's controller objects and communicate user-initiated changes - for example, text entered in a text field - through controller objects to an application's model objects.

Design patterns: Model-View-Controller (MVC)

- Controller Objects
  - acts as an intermediary between one or more of an application's view objects and one or more of its model objects.
  - Controller objects conduit through which view objects learn about changes in model objects and vice versa.
  - Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.
- A controller object interprets user actions made in view objects and communicates new or changed data to the model layer. When model objects change, a controller object communicates that new model data to the view objects so that they can display it.

Design patterns: Model-View-Presenter (MVP)

- Model View Presenter (MVP) is derived from MVC pattern.
- MVP is somewhat able to minimize high dependency on View as it separates view layer from business logic layer by presentation layer. Presentation layer decides what to be displayed on view.
  - View - renders information to users and contains UI Component .xml file, Activity, fragments, Dialog. Views are what's displayed to the user. They also handle any interaction a user may have with the screen (for example click listeners). It should not contain any business logic.
  - Model - plays role of domain or business layer and is data source of pattern. It describes the main logic of application and decides from where the data should be fetched.
  - Presenter - performs the functionality of Controller and act as middle layer between view and model layer. But unlike controller, it is not much dependent on View. View contacts presenter for the data to be presented, then Presenter takes data from model and returns to view in presentable format. Presenter is, for example, a simple java class (interface) that do not contain any UI components, it just manipulates data from model and displays it on view.

Design patterns: MVC/MVP

- Why use MVC/MVP?
  - Models and views become reusable
  - Separating code makes testing easier
  - App is easier to maintain and extend over time

- MVVM

- Other design patterns?

Design patterns: Delegation

- Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object when that object encounters an event in a program.
  - The delegating object keeps a reference to the other object - the delegate - 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 application, and in some cases it can return a value that affects how an impending event is handled.
- The main value of delegation is that it allows us to easily customize the behavior of several objects in one central object.

Design patterns: Delegation

- We can also define Delegation as – objects talking to objects!
- Delegation is implemented using
  - "protocols" and "delegates" in Swift
  - "interfaces" and "implements" in Java

Android Application Architecture

- An Android application consists of many "moving parts" whose natures and interactions need to be understood in order to develop effectively.
- An Android application consists of one or more of the following components, written as Java or Kotlin classes:
  - **Activity**
  - **Service**
  - **Broadcast receivers**
  - **Content providers**

Android – Application Components

- Activity - is a single, standalone module of application functionality that usually correlates directly to a single user interface screen and its corresponding functionality
  - An Activity comprises the visual components ("views") for one screen as well as the code that displays data into that screen and can respond to user events on that screen.
  - Activities are created as subclasses of the Android Activity class and must be implemented so as to be entirely independent of other activities in the application.
  - One activity cannot directly call methods or access instance data of another activity.

- Fragment - is a section of an activity
  - Each fragment (subclass of the Android Fragment class) consists of part of the user interface layout and a matching class file.
  - A Fragment can only exist inside an Activity.
  - A single activity can switch between different fragments, each representing a different app screen.

Android – Application Components

- **Service** - is a component that runs in the background to perform long running operations or to perform work for remote processes. A service does not provide a user interface.
  - Although services lack user interface, they can still notify the user by providing notifications and toasts.
  - Services can issue intents.
  - Service can be declared to run in the foreground - only for situations where termination would be detrimental to the user experience.
  - For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.

Android – Application Components

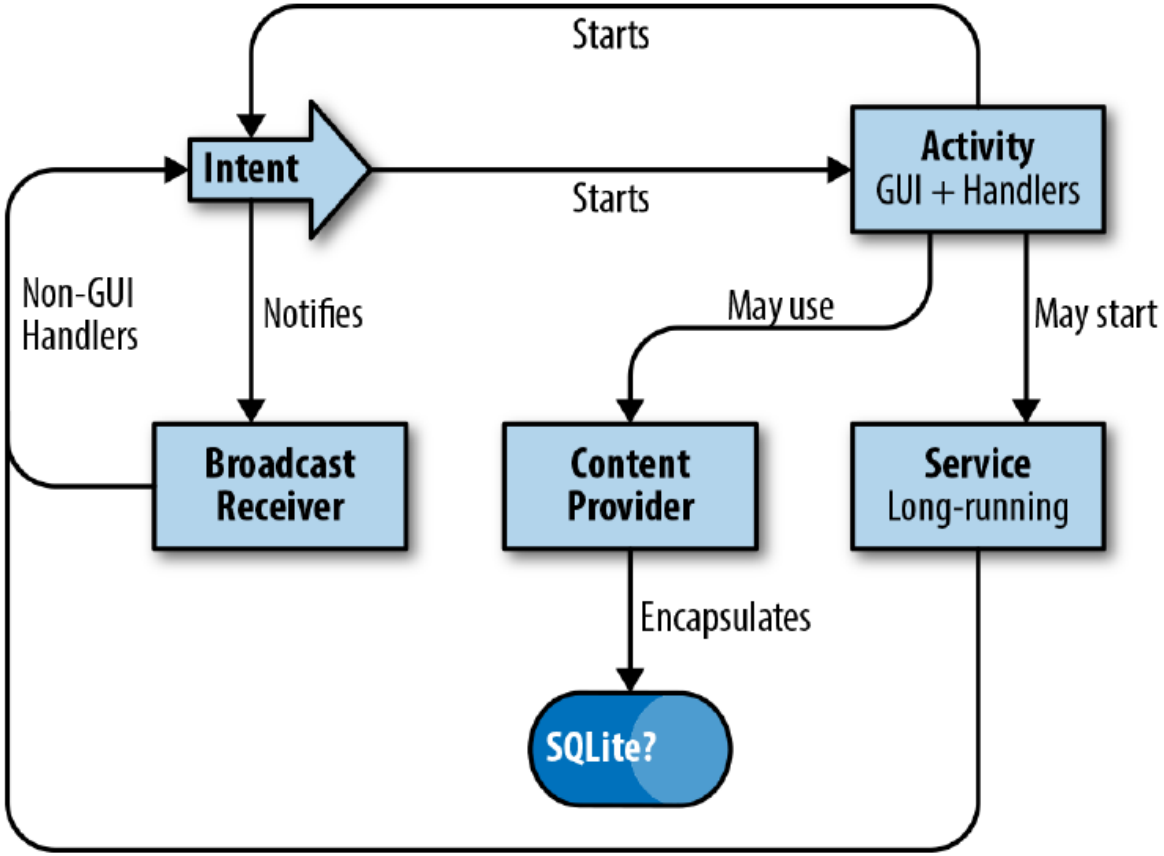
- **Broadcast Receiver** - is a component that enables the system to deliver events to the app outside of a regular user flow.
  - Many broadcasts originate from the system - for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
  - It allows the app to respond to system-wide broadcast announcements.
  - Apps can also initiate broadcasts - for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
  - Broadcast receivers don't display user interface, they can create status bar notifications.

- **Content providers** are used when one application needs to share its data with other applications.

- **Sync adapters** synchronize data with cloud services; the best-known examples are the Contacts and Calendar apps on the device, which can easily be synchronized to your Google account.

Android – Activating Components

- Creation of these objects requests using an ***Intent***, an object that specifies your intention to have something done.
- Intents can start Activities within our application (by class name), start Activities in other applications (by specifying type of action and other information), start Services or Broadcast receiver, and request other operations.



Source: Android Cookbook (2nd edition), Ian Darwin, O'Reilly Media, 2017.

Android – Activating Components

- **Activities**, **services**, and **broadcast receivers -** are activated by an **Intent**.
- **Content provider**, is not activated by intents. Rather, it is activated when targeted by a request from a ContentResolver.
  - The content resolver handles all direct transactions with the content provider. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

Android – Intents and Intent Filters

- An **Intent** is a messaging object used to request an action from another app component.
- There are three fundamental use-cases:
  - To start an activity - a new instance of an Activity can be started by passing an Intent to startActivity();
  - To start a service - a service can be started by passing an Intent to startService();
  - To deliver a broadcast - the system delivers various broadcasts for system events (for example when the system boots up or the device starts charging).

Android – Intents and Intent Filters

- **Explicit intents** specify the component to start by name (the fully-qualified class name).
  - Typically is used to start a component with-in the app.
  - For example, start a new activity in response to a user action or start a service to download a file in the background.

- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
  - For example, to take a photo, the app can use an implicit intent to request another app to take the photo.

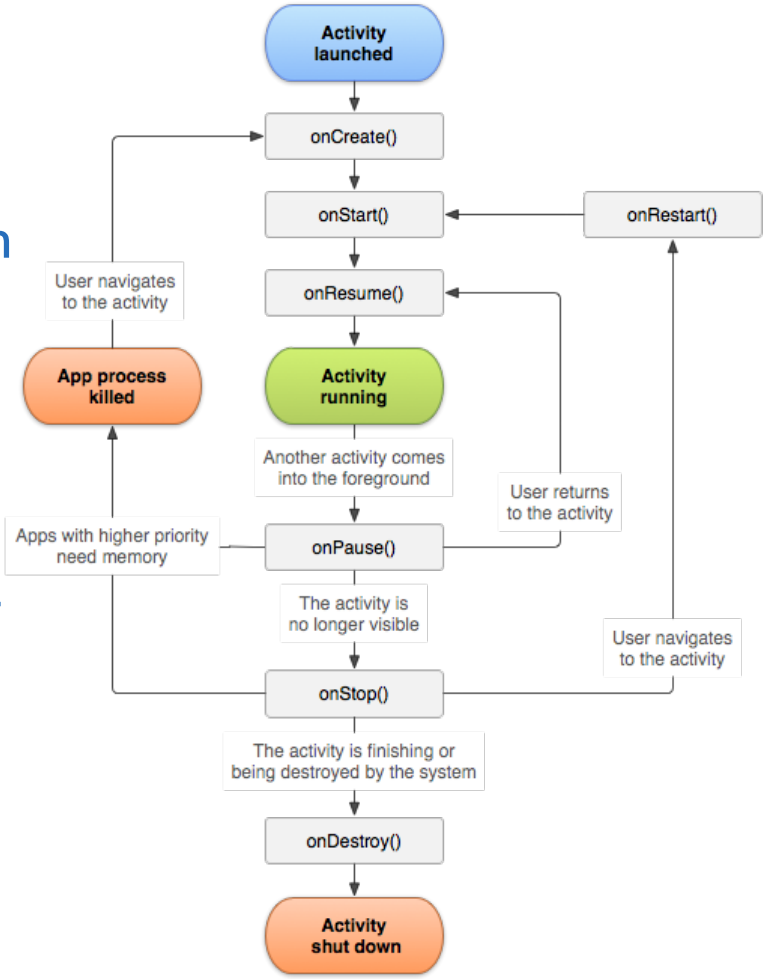
Android – Intents and Intent Filters

- An **intent filter** is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
  - By declaring an intent filter for an activity, it is possible for other apps to directly start that activity with a certain kind of intent.
  - If the intent filter(s) is not declared for an activity, then that activity can be started only with an explicit intent.
- The power of intents lies in the concept of implicit intents – it allows the system to find a component on the device that can perform the action and start it.
  - If there are multiple components that can perform the action described by the intent, then the user selects which one to use.

***Caution**: For security always use an **explicit intent** when starting a Service and do not declare intent filters for your services. Beginning with Android 5.0 (API level 21), the system throws an exception if you call bindService() with an implicit intent.*

Android Activity Life Cycle

- Activity is the most basic component of Android application.
- The class *android.app.Activity* provides a number of well-defined life-cycle methods that are called when an application is started, suspended, restarted, ...
- An Android app can be in one of three states:
  - **Active** - in which the app is visible to the user and is running;
  - **Paused** - in which the app is partly obscured and has lost the input focus (e.g. when a dialog is in front of app's Activity);
  - **Stopped** - in which the app is completely hidden from view.



Android apps do not have a "main" method

Source: https://developer.android.com/guide/components/activities/activity-lifecycle

Android – The Manifest File

- Before the Android system can start an app component, the system must know that the component exists by reading the app's *AndroidManifest.xml file*.
- The app must declare all its components in this file, which must be at the root of the app project directory.
- The manifest does a number of things in addition to declaring the app's components, such as:
  - Identify any user permissions the app requires, such as Internet access or read-access to the user's contacts.
  - Declare the minimum API Level required by the app.
  - Declare hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multitouch screen.
  - API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
  - …

Android – App Resources

- An Android app requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the app.
  - For example, animations, menus, styles, colours, and the layout of activity user interfaces defined with XML files.
- Using app resources makes it easy to update various characteristics of the app without modifying code.
- For every resource, the SDK build tools define a unique integer ID, which can be used as reference to the resource from the app code or from other resources defined in XML.
  - For example, if the app contains an image file named appimage.jpg (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.appimage, which can be used as reference to the image and insert it in the user interface.

Android – App Resources

- **strings.xml** is the default resource file used to hold name/value pairs of strings so that they can be referenced throughout the app.

Source: Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017.

Android – Application Fundamentals

- Android apps are written in the Java or Kotlin programming language.
- The Android SDK tools compile app's code along with any data and resource files into an APK: an Android package, which is an archive file with an .apk suffix.
- One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.
- Once installed on a device, each Android app lives in its own security sandbox:
  - The Android operating system is a multi-user Linux system in which each app is a different user.
  - Each app has a unique Linux user ID (used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
  - Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
  - Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

Android – Application Fundamentals

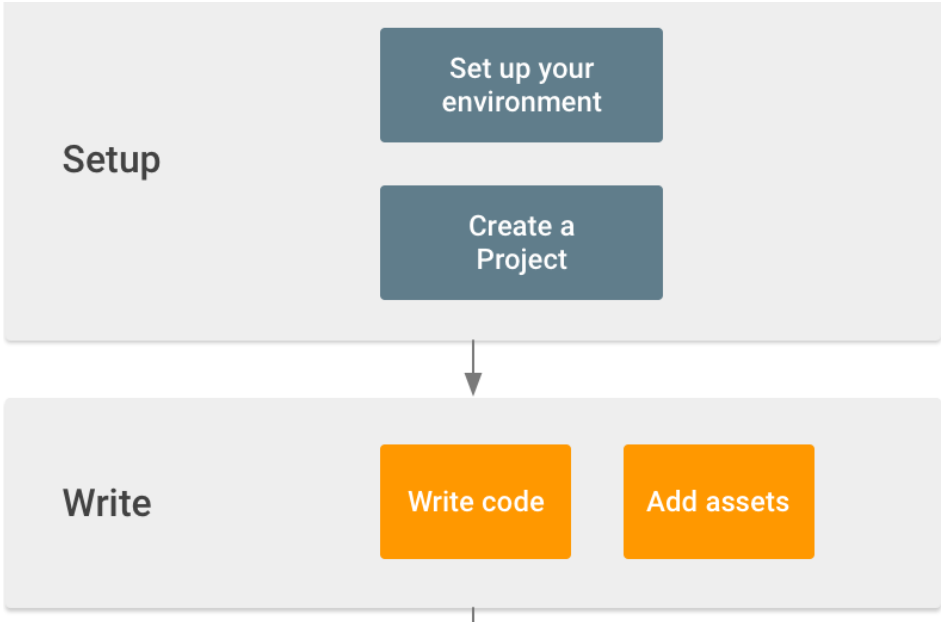
- Security of Android Apps:
  - Android system implements the principle of least privilege - each app has access only to the components that it requires to do its work and no more.
  - An app cannot access parts of the system for which it is not given permission.
- An app can share data with other apps and can access system services:
  - If two apps share the same Linux user ID - they are able to access each other's files.
  - The apps with the same user ID can also arrange to run in the same Linux process and share the same VM.
  - An app must request permission to access device data (user's contacts, SMS messages, SD card, camera, Bluetooth, …).

Android – Declaring app requirements

- There are a variety of devices powered by Android and not all of them provide the same features and capabilities.
- The profile for the types of devices and software requirements should be declared in the manifest file - it will prevent the app to be installed on devices that lack needed features.
- Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read it in order to provide filtering for users when they search for apps from their device.

Android Application Development Workflow

- Setup
  - ✓ setting up your Android Studio and install SDK
  - ✓ Creating AVD – target device

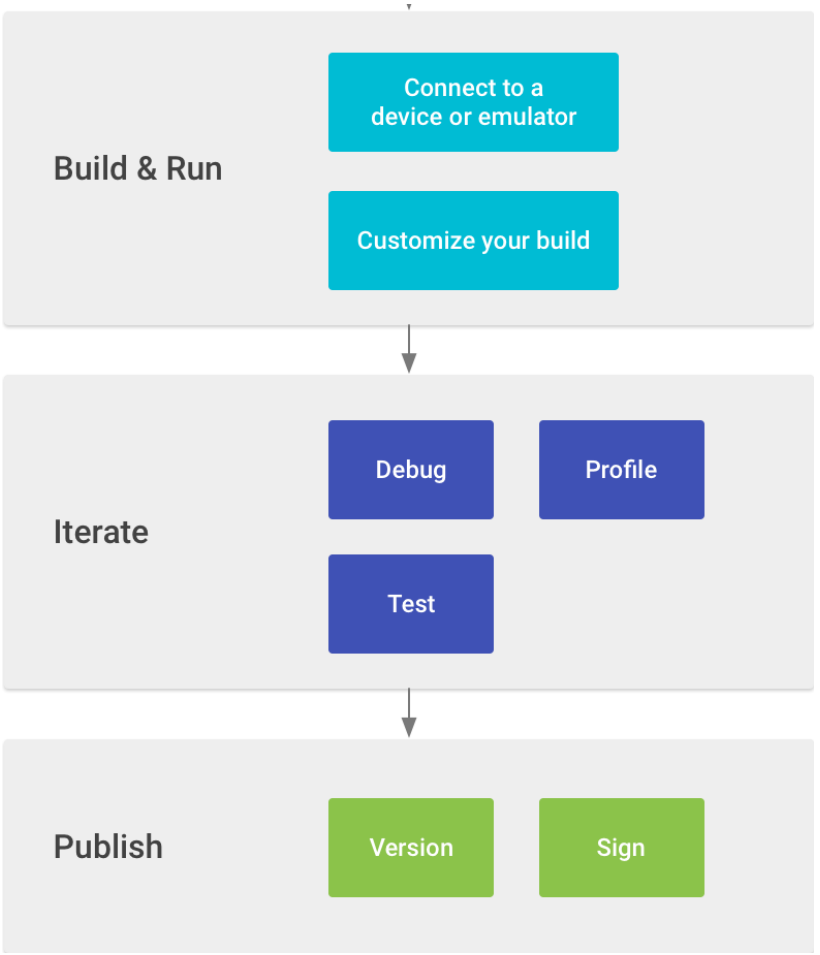


- Development / Write
  - ✓ Creating application
  - ✓ Layout and code/activities
  - ✓ Deciding on services to use

Source: http://www.androiddocs.com/tools/workflow/index.html

Android Application Development Workflow

- Build and run / Debugging and Testing
  - ✓ Use of AVD and debugging tools
  - ✓ Testing and performance tune
- Publishing
  - ✓ Prepare for release
  - ✓ Maintenance and decision for marketing…



Source: http://www.androiddocs.com/tools/workflow/index.html

Android APIs

- **API Level** is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.
- The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:
  - A core set of packages and classes,
  - A set of XML elements and attributes for declaring a manifest file,
  - A set of XML elements and attributes for declaring and accessing resources,
  - A set of Intents,
  - A set of permissions that applications can request, as well as permission enforcements included in the system.

iOS App and Environment

- During startup, the *UIApplicationMain* function sets up several key objects and starts running the app:
  - creating the core objects of the app,
  - loading app's user interface (from the available storyboard files),
  - calling custom code so some initial setup can be done,
  - putting the app's run loop in motion.

- In iOS 13 and later, the user can create and manage multiple instances of the app's user interface simultaneously
  - switch between them using the app switcher, or
  - on iPad, multiple instances of the app can be displayed side by side.
- Each instance of the UI displays different content or displays the same content in a different way.

iOS apps do have a "main" method in Objective-C or UIApplicationMain function - @UIApplicationMain special attribute in Swift.

iOS App and Environment Lunching the app

In iOS 12 and before

- UIApplicationMain:
  - instantiates UIApplication that serve as the shared application instance;
  - instantiates the app delegate class - the application instance's delegate;
  - If the app uses a main storyboard
    - ✓ instantiates the main storyboard's initial view controller,
    - ✓ instantiates UIWindow (assigns the window instance to the app delegate's *window* property),
    - ✓ assigns the initial view controller instance to the window's *rootViewController* property;
  - calls the app delegate's *application(_:didFinish-LaunchingWithOptions:);*
  - calls the window's instance method *makeKeyAndVisible* (to make the app's interface visible).

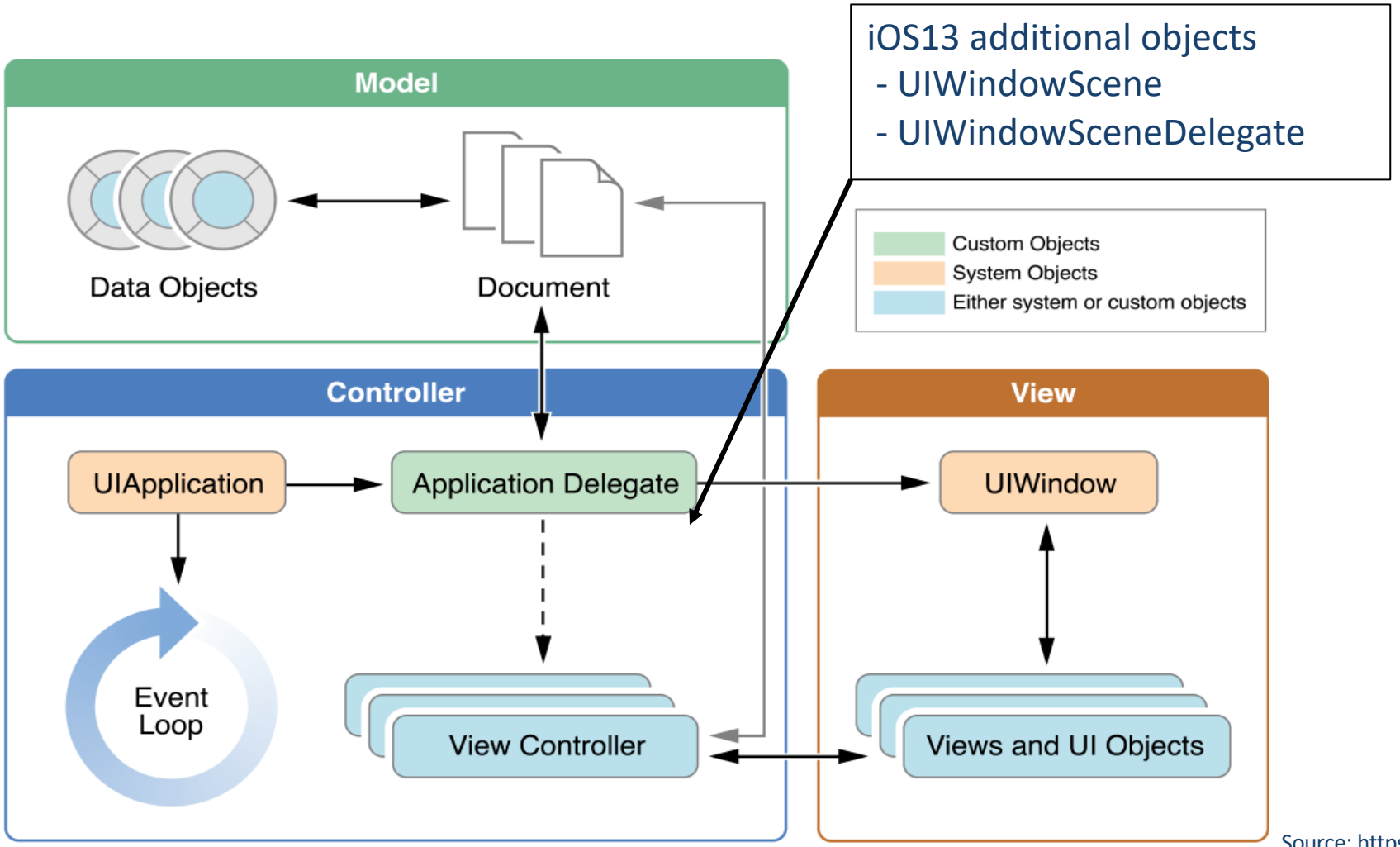
In iOS 13 and later

- UIApplicationMain
  - instantiates UIApplication that serve as the shared application instance;
  - instantiates the app delegate class - the application instance's delegate;
  - calls the app delegate's application(_:didFinish-LaunchingWithOptions:);
  - creates a UISceneSession, a UIWindowScene, and an instance that will serve as the window scene's delegate;
  - If the initial scene uses a storyboard:
    - ✓ instantiates that storyboard's initial view controller,
    - ✓ instantiates UIWindow (assigns the window instance to the scene delegate's *window* property),
    - ✓ assigns the initial view controller instance to the window instance's *rootViewController* property;
  - calls the window's instance method *makeKeyAndVisible* (to make the app's interface visible).
- The scene delegate's *scene(_:willConnectTo:options:)* is called.

iOS App and Environment Lunching the app

- Since Swift 5.3, @UIApplicationMain was deprecated
- Current approach:
  - Xcode automatically generates the equivalent functionality using the @main attribute
  - No need to worry about manually calling UIApplicationMain
- Where to find relevant code
  - Find the file named ”AppDelegate.swift" in your project

iOS App's objects



Source: https://developer.apple.com

iOS App's objects

- Every iOS app has exactly one instance of ***UIApplication*** *object* that
  - handle the initial routing of incoming user events,
  - dispatches action messages forwarded to it by control objects to appropriate target objects,
  - maintains a list of open windows (UIWindow objects) and through those can retrieve any of the app's UIView objects.
- The app delegate object (protocol ***UIApplicationDelegate***) is the root object of the app that manages app's shared behaviors:
  - initialize app's central data structures,
  - configure app's scenes,
  - respond to notifications originating from outside the app (e.g. low-memory warnings),
  - respond to events that target the app itself, and are not specific to the app's scenes, views, or view controllers,
  - register any required services at launch time (e.g. Apple Push Notification service).

iOS App's objects

- **UIWindow** work with app's view controllers to handle events and to perform many tasks that are fundamental for app's operation:
  - Provide a main window to display app's content,
  - Create additional windows (as needed) to display additional content.
- A **UIWindowScene** object manages one instance of the app's UI, including one or more windows that are displayed from that scene:
  - manages the display of the windows on the user's device, and the life cycle of that scene as the user interacts with it.
- **UIWindowSceneDelegate** object
  - manage the life cycle of one instance of the app's user interface,
  - respond to changes in the underlying environment of the scene.
- **UISceneDelegate** object
  - manage life-cycle events in one instance of the app's user interface,
  - defines methods for responding to state transitions that affect the scene, including when the scene enters the foreground and becomes active, and when it enters the background.

iOS App's objects

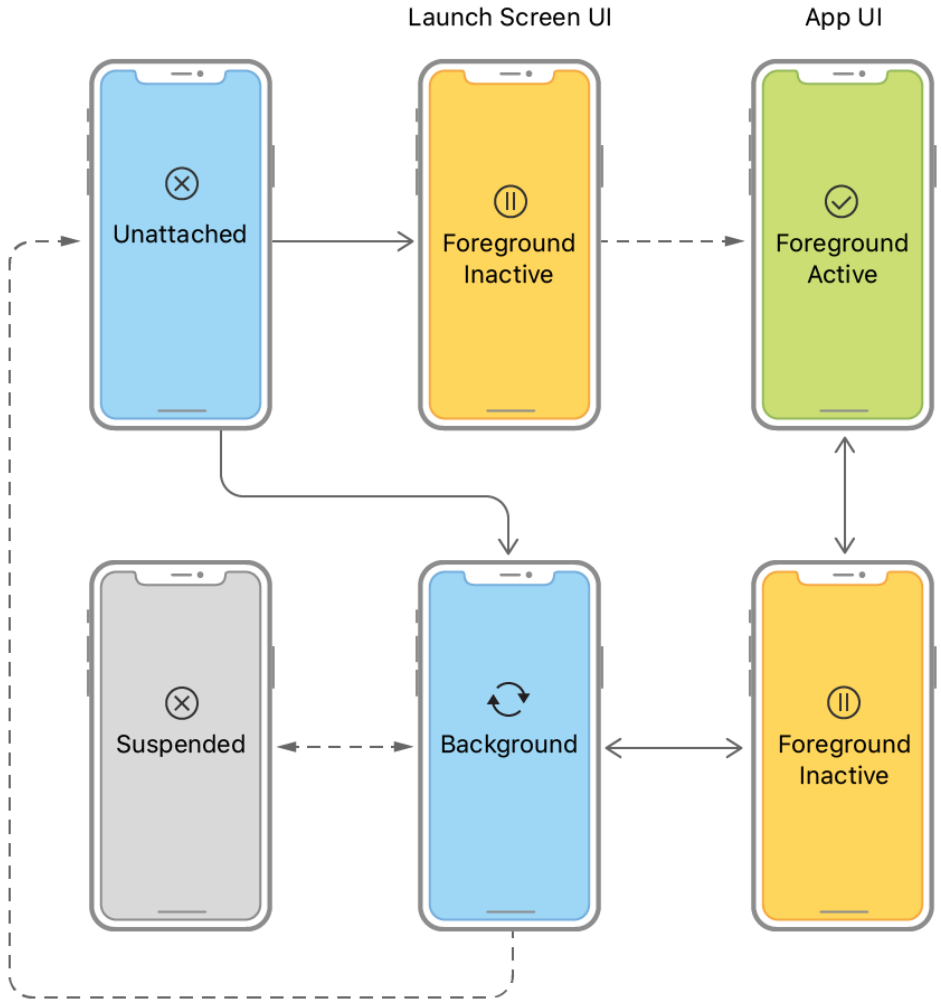
- **Data model objects** store app's content and are specific for every app.

- Apps can also use **Document objects** (custom subclasses of UIDocument) to manage discrete portions of app's data
  - Document objects are not required but offer a convenient way to group data that belongs in a single file or file package.

- **View Controller** objects manage the presentation of the app's content on screen
  - **Views** and **controls** provide the visual representation of your app's content.

iOS - App's Life Cycle

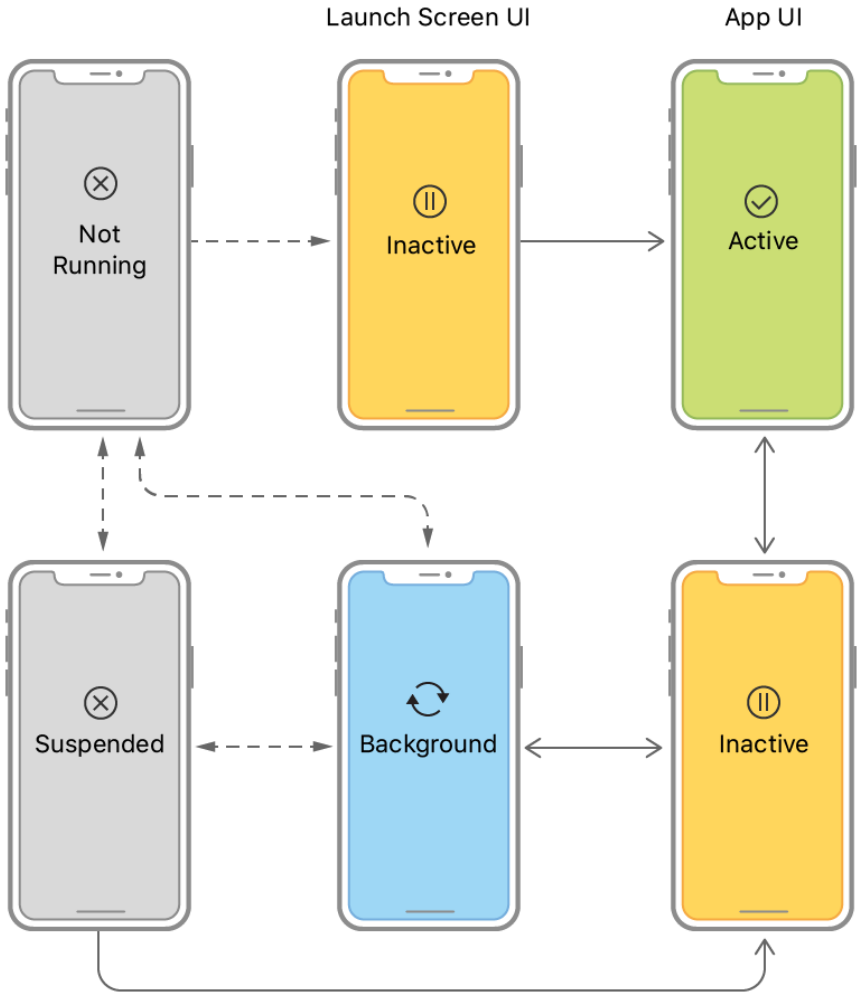
- Scene-Based Life-Cycle Events
  - If the app supports scenes - UIKit delivers separate life-cycle events for each.



Source: https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle

iOS - App's Life Cycle

- App-Based Life-Cycle Events
  - In iOS 12 and earlier, and in apps that don't support scenes - UIKit delivers all life-cycle events to the UIApplicationDelegate object.



Source: https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle

iOS App resources and metadata

- Every app must have the following set of resources and metadata so that it can be displayed properly on iOS devices:
  - An information property-list file - The *Info.plist* file contains metadata about the app, which the system uses to interact with the app. (Xcode creates this file automatically based on the project's configuration and settings.)
  - A declaration of the app's required capabilities - Every app must declare the hardware capabilities or features that it requires to run. The App Store uses this information to determine whether or not a user can run the app on a specific device.
  - One or more icons - The system displays the app icon on the home screen of a user's device.
  - Launch Screen - When an app is launched, the system displays a temporary lunch screen until the app is able to present its user interface.

When you build your iOS app, Xcode packages it as a bundle. A bundle is a directory in the file system that groups related resources together in one place. An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.

Resources

- Head First Android Development, Dawn Griffiths and David Griffiths, O'Reilly Media, 2017
- iOS 17 App Development Essentials, Neil Smyth, Payload Media, Inc., 2023.
- iOS 17 Programming for Beginners - 8th Edition, Ahmad Sahar, Packt, 2023.
- https://developer.android.com/guide/platform
- https://developer.apple.com/ios/
- https://intellipaat.com/tutorial/ios-tutorial/
- https://www.android.com/android-14/
- https://www.apple.com/au/ios/ios-17/features/

Resources

- Android Studio 4.1 Essentials - Java Edition, Neil Smyth, Payload Media Inc., 2020.
- Android Cookbook (2nd edition), Ian Darwin, O'Reilly Media, 2017.
- Programming iOS 13: Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2020.
- Programming iOS 14: Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2020.
- iOS 17 App Development Essentials, Neil Smyth, Payload Media, Inc., 2023.
- https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html
- https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html
- https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle
- https://developer.android.com/guide/components/fundamentals
- http://www.androiddocs.com/tools/workflow/index.html