

Mobile Applications for Sensing and Control

Week 1

Sep Makhsous

Teaching Team

- **Instructor:**

- Sep Makhsous, sosper30@uw.edu
 - Office hours: By appointment

- **TA:**

- Tejashree Nitin Khedkar
- Manyu Chen

Class Time and Location

- T 6:00 pm-10:00 pm, ECE 045
- In Person Class
 - Offered via Zoom (Limited Interactions)

Course Structure

- Lecture Format:
 - Some Lecture
 - In class working sessions and exercises
- Prereq
 - Java or similar programming language
- Course Elements and grading weight:
 - Homework: 50%
 - ICTEs: 10%
 - Final Project: 40%
 - Bonus: 5%

Homework Assignments (50%)

- Individual
- 3 Homework assignments
- Due on Saturday at 11:59 PM (for 5% bonus)
 - Late submissions accepted until Sunday at 11:59 PM – Full credit but no bonus

In Class Team Exercises (ICTEs) 10%

- Teams of 2
- At least 4 ICTEs overall
- Lowest ICTE will be dropped
- Due at the end of the class
- You need to include all Authors names when you submit your work

Final Project (individual) 40%

- Objective:
 - The goal of this final project is to demonstrate the skills and knowledge learned in this course by developing a mobile application using React Native that integrates with native APIs and potentially external APIs.
- You pick a topic from a suggested list.
- The final project should demonstrate the ability to access native APIs and to communicate with external APIs if necessary
- Final Projects are due on June 8th at midnight

About the Class

- In this course, students will gain the practical skills necessary to develop modern mobile applications, taking advantage of many sensors and control capabilities that modern smartphones offer. Students will use Kotlin, an open-source framework to create native mobile apps for Android.

Learning Outcomes

By the end of this course, you will be able to:

- Apply Kotlin programming concepts to develop Android mobile applications.
- Construct dynamic user interfaces using Android's rich set of layout managers and view components.
- Utilize Android's extensive sensor and control APIs for building interactive applications.
- Manage user interactions and navigate between different activities in an Android app.

Canvas Demo

What is Android

- Open-source OS and development platform
- In theory, you can change anything (in practice, it's a bit more complicated)
- Hardware reference design, Linux OS kernel
- Open-source libraries for app development (e.g., SQLite, Webkit, OpenGL, media manager)
- Uses Java but is compiled to bytecode for small, efficient, and multiple VMs
- Application framework, UI framework, preinstalled apps
- SDK and tools, wild west of app stores: The Play Store

Why Android Apps?

- Better UI and user experience compared to web pages
- More direct access to device hardware (e.g., camera, GPS)

Why not iOS?

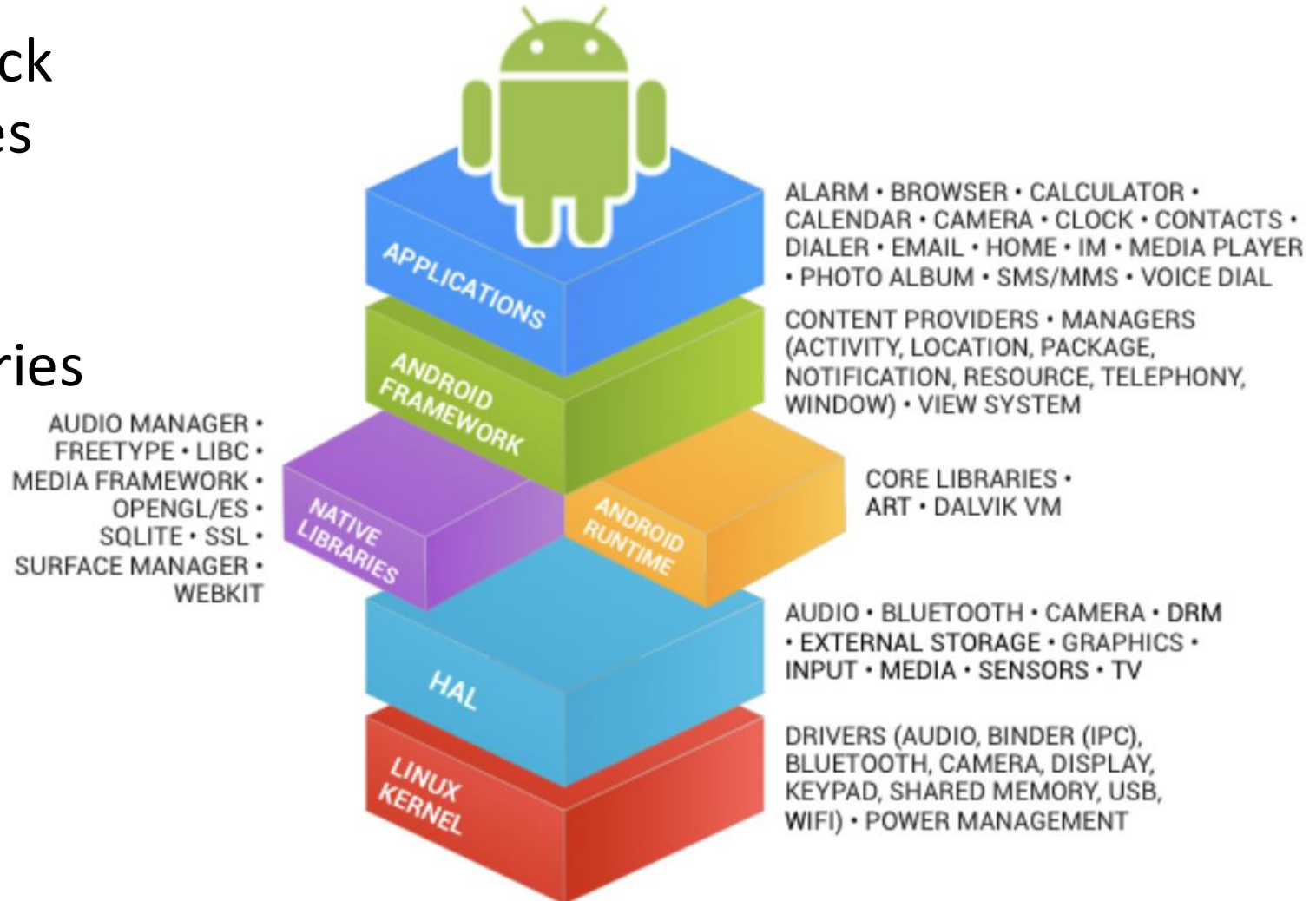
- Familiar language (Java/Kotlin) vs Swift/Objective-C
- Free development tools, easy to install apps on devices
- Larger install base
- We have a bunch of Android phones

Hybrid Mobile App Frameworks

- Cross-platform apps with:
 - React Native
 - Xamarin
 - Ionic
 - Flutter

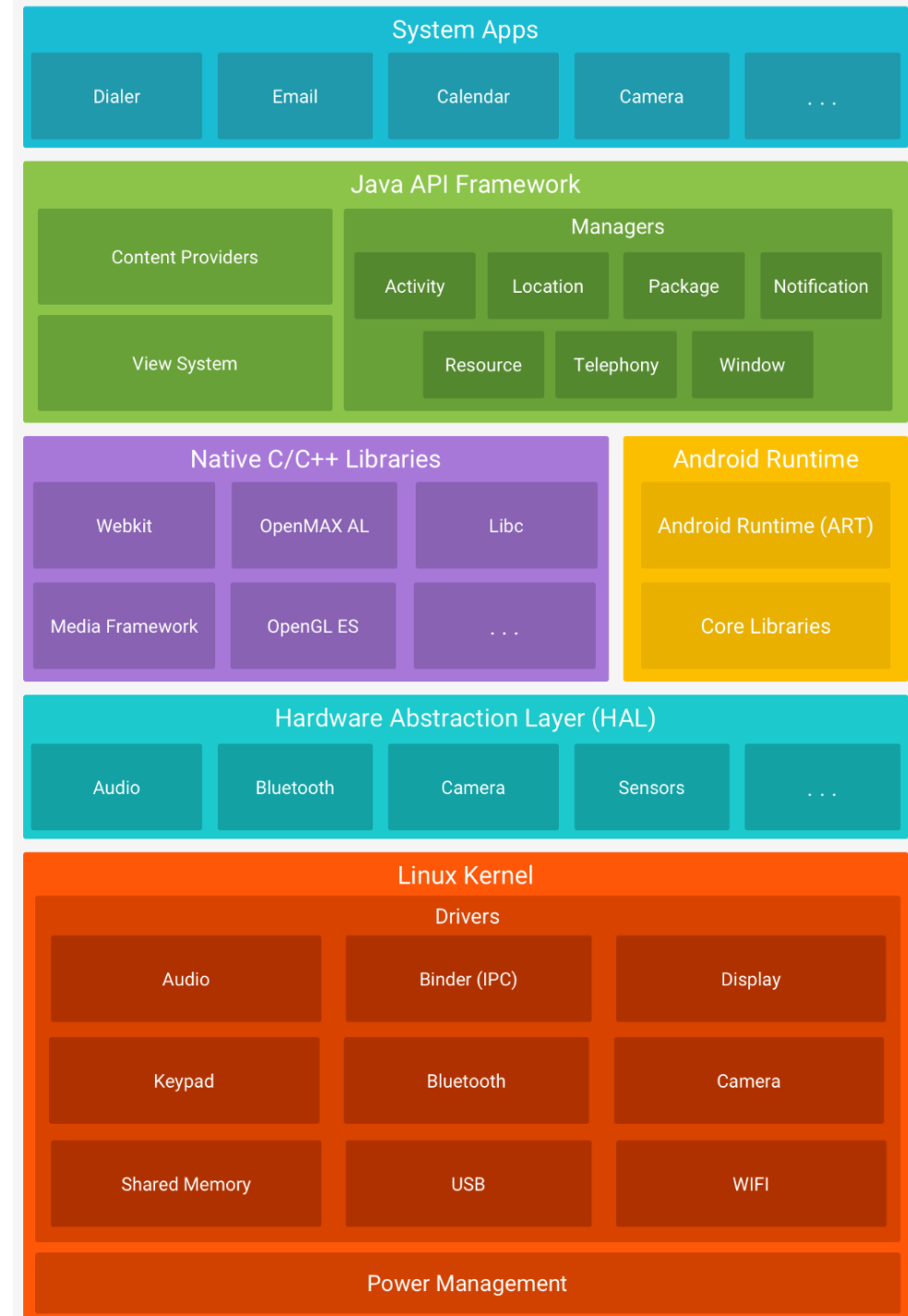
System Architecture

- Linux-based software stack for a wide array of devices and form factors
- Features of the devices used/accessed with libraries



System Architecture

- Linux-based software stack for a wide array of devices and form factors
- Features of the devices used/accessed with libraries



Notables

- Open philosophy, encourages reuse of application components
- Access to much hardware (sometimes even without bugs!)
- Release and fix mentality
- Great standard apps (e.g., Google Maps, location services, quick search box)
- Background services

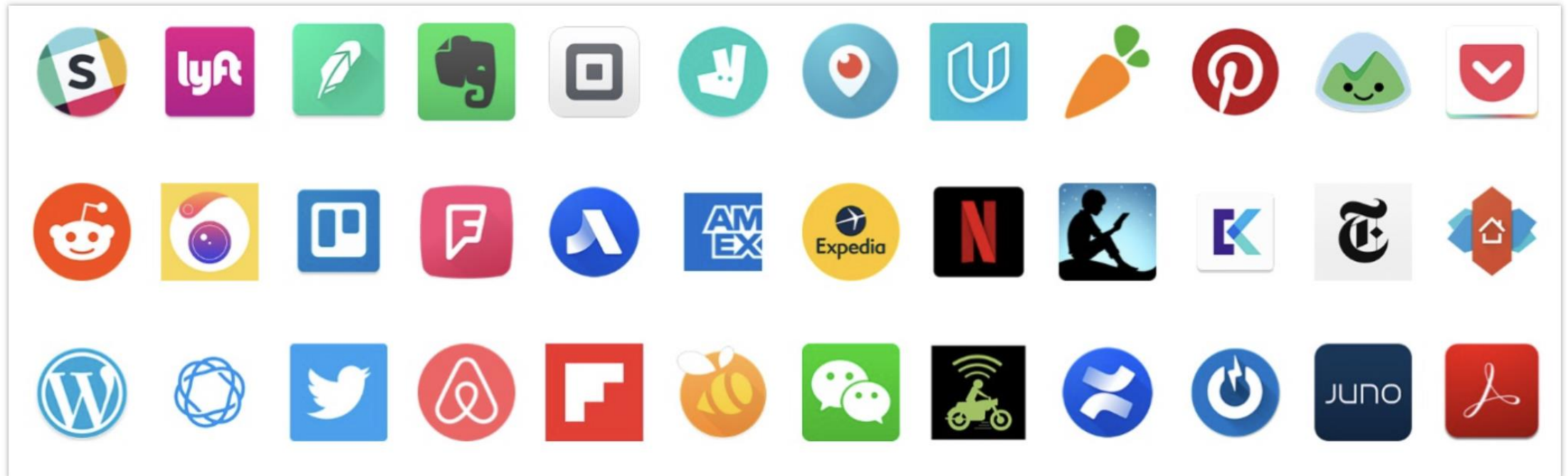
Intro to Kotlin

- Android apps can be written using Kotlin, Java, and C++ languages
- Java: legacy code; Kotlin: first-class support by Google (since 2017); C++: for digital signal processing

Kotlin Overview

- Developed by JetBrains in 2011, first-class language for Android Apps
- Announced at Google I/O 2017
- Advantages include concise syntax, modern features, and seamless interoperability with legacy Java code.

Who uses Kotlin



Why Kotlin

- Official language recommended by Google
- Cleaner, safer, and easier than Java
- Modern, concise — great for beginners
- Example comparison:

```
// Java  
String name = "World";
```

VS

```
// Kotlin  
val name = "World"
```

Variables

- Variables in Kotlin, as in Java, allow us to assign values that can then be modified and used at different points in our program as long as they are within the scope in which the code is executed.

Java

Need to specify type:
Int, float, double, String

// Constants

```
final double d = 1.0;
```

Kotlin

Use **var**->type is inferred

// Constants (read only)

Use **val**->type is inferred

Variables

- In Kotlin, types are inferred automatically, so you don't have to write them unless you want to.
- `val` = immutable (like `final` in Java) → safer by default.
- `var` = mutable.

Java

```
String name = "Android";  
int age = 5;
```

Kotlin

```
val name = "Android"  
var age = 5
```

String Concatenation

- Kotlin has built-in string templates, which are cleaner and easier to read
- You can even write expressions inside them:

kotlin

```
"You are ${age + 1} years old"
```

java

```
String greeting = "Hello, " + name + "!";
```

kotlin

```
val greeting = "Hello, $name!"
```


Null Safety

- Kotlin enforces null safety in its type system.
- String? means it can be null.
- The ? operator safely handles null values without crashing the app.

```
String text = null;  
int length = text.length(); // ⚠️ NullPointerException!
```

```
val text: String? = null  
val length = text?.length // ✅ Safe call; returns null if text is null
```

Less Boilerplate Code

- In one line, Kotlin generates:
 - Constructor
 - Getters
 - toString(), equals(), hashCode()...
- This saves tons of boilerplate code in real apps

Less Boilerplate Code















java

```
public class User {  
    private String name;  
  
    public User(String name) {  
        this.name = name;  
    }  
  
    public String getName() { return name; }  
  
    public void setName(String name) { this.name = name; }  
}
```

Other Benefits

- **when** instead of **switch**
 - Kotlin's when expression replaces Java's clunky switch with a cleaner, more flexible syntax — works with any type, not just integers.
- Lambdas for Event Handling
 - No more anonymous classes! Kotlin uses lambdas for clean, readable UI interactions
- Extension Functions
 - Add new functionality to existing types without modifying their source code — perfect for cleaner utilities and reusable code.

Comparison

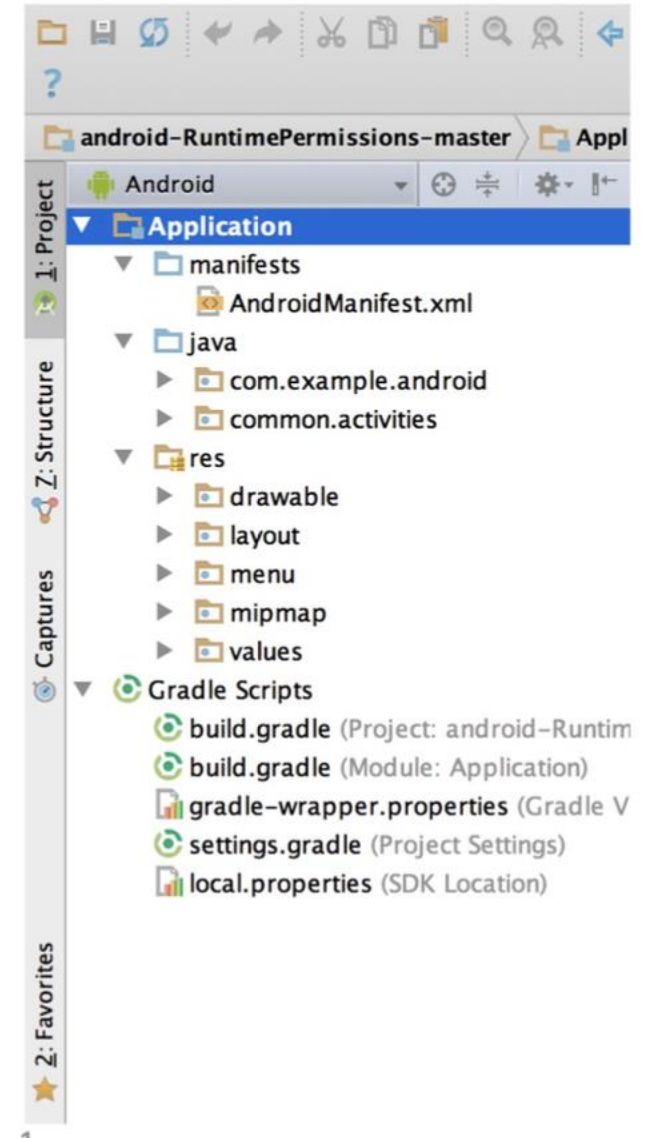
Feature	Kotlin	Java
Type inference	 Yes	 No
Null safety	 Yes	 No
Data classes	 1 line	 Lots of boilerplate
Lambdas	 Modern and concise	 Verbose
String templates	 Yes	 No
Extension functions	 Yes	 No
Safety for beginners	 High	 Prone to NullPointerExceptions

Android Studio

- Official IDE for Android development
- Replaces previous Eclipse environment, based on IntelliJ IDEA editor
- Free download and use: <https://developer.android.com/studio/>


Project Structure

- Each app module contains:
 - manifests: AndroidManifest.xml file
 - java: Java source code files, including JUnit test code
 - res: Non-code resources, such as XML layouts, UI strings, and bitmap images



Project Structure

man·i·fest²

/ˈmanəˌfest/ 

noun

noun: **manifest**; plural noun: **manifests**

1. a document giving comprehensive details of a ship and its cargo and other contents, passengers, and crew for the use of customs officers.
 - a list of passengers or cargo in an aircraft.
 - a list of the cars forming a freight train.

verb

verb: **manifest**; 3rd person present: **manifests**; past tense: **manifested**; past participle: **manifested**;
gerund or present participle: **manifesting**

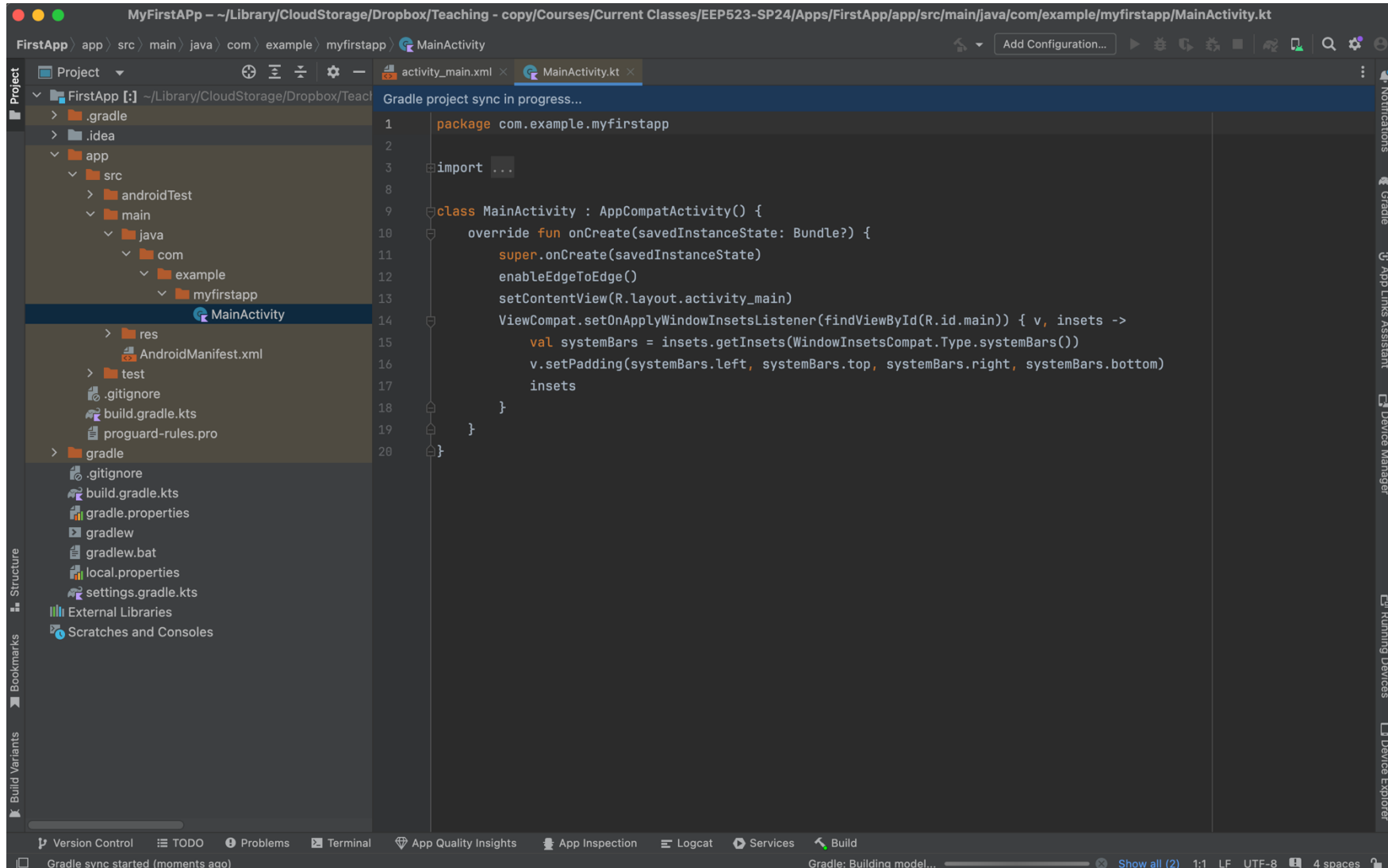
1. record in a manifest.
"every passenger is manifested at the point of departure"

Parts of a Project: Manifest

- AndroidManifest.xml tells the system what components exist
- Sets intent filters, permissions, minimum API Level (now in Gradle)
- Declares hardware & software features used or required
- Specifies link to API libraries other than the Android framework APIs (e.g., Google maps)
- Sets icons

Building your first android App

Android Studio



Kotlin

The screenshot shows a web browser displaying the Kotlin documentation page for 'Hello world'. The browser's address bar shows the URL `kotlinlang.org/docs/kotlin-tour-hello-world.html`. A purple banner at the top of the page reads 'Shape the future of build configurations by sharing your insights on a new language →'. The page has a dark navigation bar with the Kotlin logo (v1.9.23) and links for Solutions, Docs, Community, Teach, and Play. A search icon is also present. On the left, a sidebar lists various topics like Home, Get started, Take Kotlin tour, Kotlin overview, What's new in Kotlin, Releases and roadmap, Basics, Concepts, Multipatform development, Platforms, Standard library, Official libraries, API reference, Language reference, Tools, Compiler and plugins, Learning materials, Early access preview (EAP), and Other resources. On the right, a vertical list of links includes Hello world, Variables, String templates, Practice, Exercise, and Next step. The main content area features the title 'Hello world' with a sub-link 'Edit page' and the text 'Last modified: 24 July 2023'. Below this is a numbered list of topics: 1 Hello world, 2 Basic types, 3 Collections, 4 Control flow, 5 Functions, 6 Classes, and 7 Null safety. The text 'Here is a simple program that prints "Hello, world!":' is followed by a code block containing the following Kotlin code:

```
fun main() {  
    println("Hello, world!")  
    // Hello, world!  
}
```

Let's Build one simple app together

- See icte 2