



Lab on apps development for tablets, smartphones and smartwatches

Week 1: **Android overview. Defining a GUI.**

Giovanni Ansaloni

Rafael Medina, Hossein Taji, Yuxuan Wang
Qunyou Liu, Amirhossein Shahbazzinia, Christodoulos Kechris

School of Engineering (STI) – Institute of Electrical and Micro Engineering (IEM)



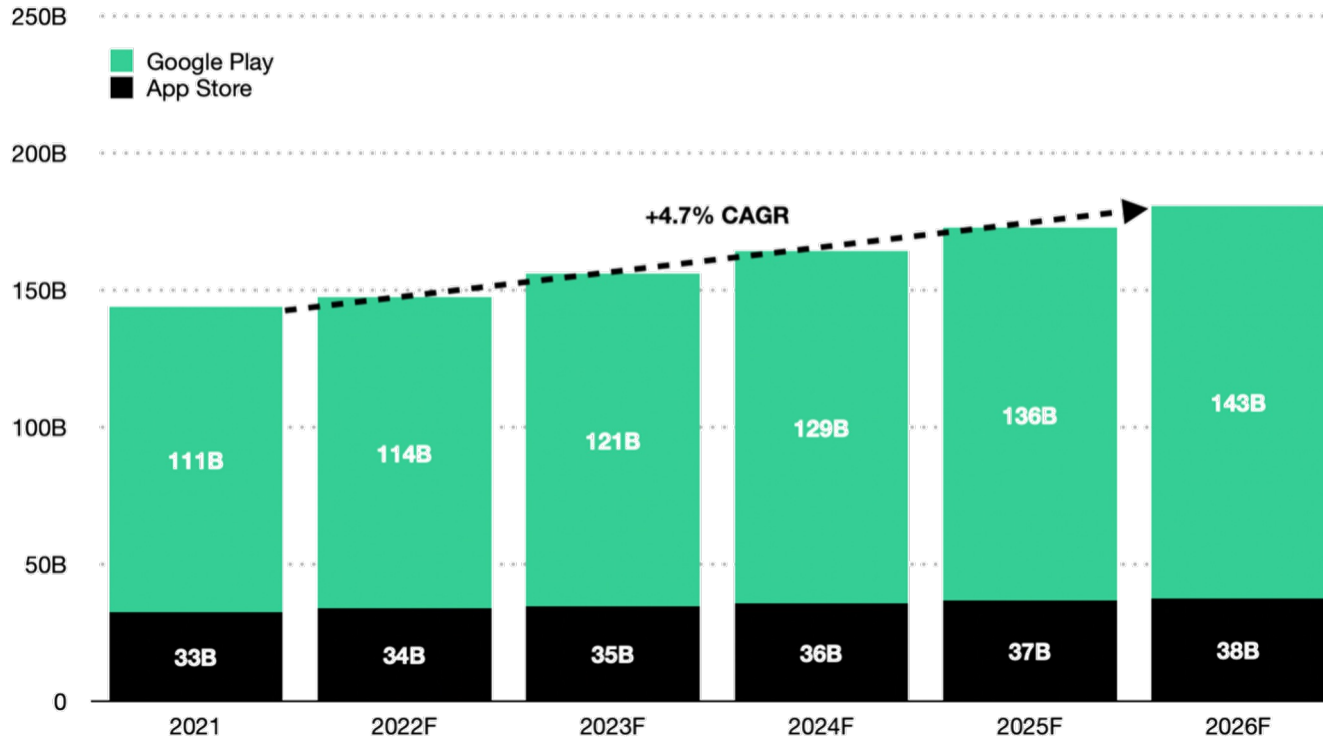
Outline of the course

0. Course presentation. Introduction to Kotlin.
1. **Android overview. Defining a GUI.**
2. Dynamic applications: State and interactivity.
3. Complex GUIs: Screens and menus.
4. Apps under the hood: Life cycles
Communication between Android and AndroidWear devices: Wear APIs.
5. Separating concerns: UI controllers and viewModels.
Interfacing with sensors: System Services.
6. Interfacing with the cloud: Firebase.
Displaying structured data: Lists.
7. Local databases: Room library.
Integrating Google maps.
8. Bluetooth Low Energy.



Growth of the mobile app economy

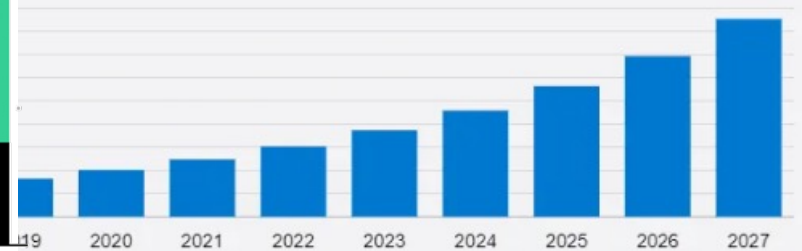
Global App Store and Google Play Installs 2021-2026



source: SensorTower, 2022

source: Technavio, 2024

Market Size Outlook (USD Billion)



2017 : USD 225.28

© ESL-EPFL

 **22.85%**
Year-over-Year
growth rate of 2023

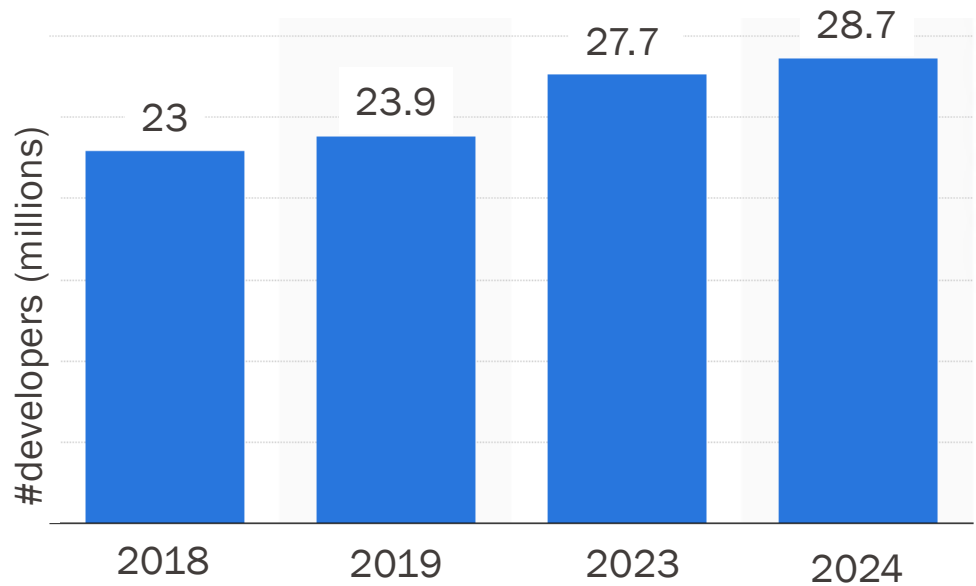
 **22.97%**
CAGR 2022-2027

 **ACCELERATING**
Growth Momentum

 **USD 1095.9 Bn**
Market size
growth

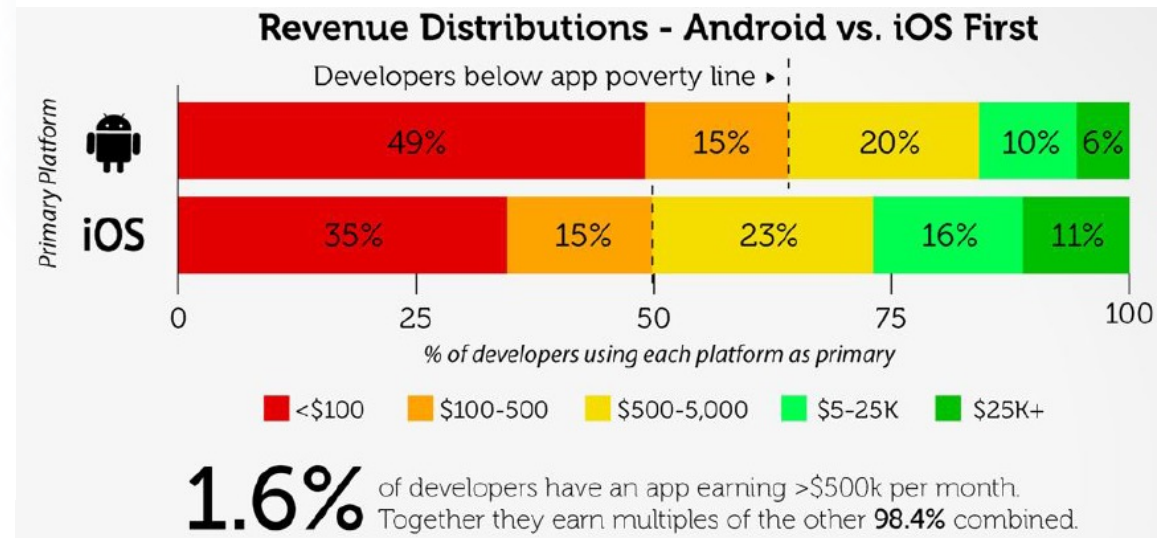
Apps market

- ~24M developers in 2019, 900K new developers per year
- 2% of developers earn 54% of all app. revenue
- 64% of Android developers are below the app. poverty line (<500\$/month)



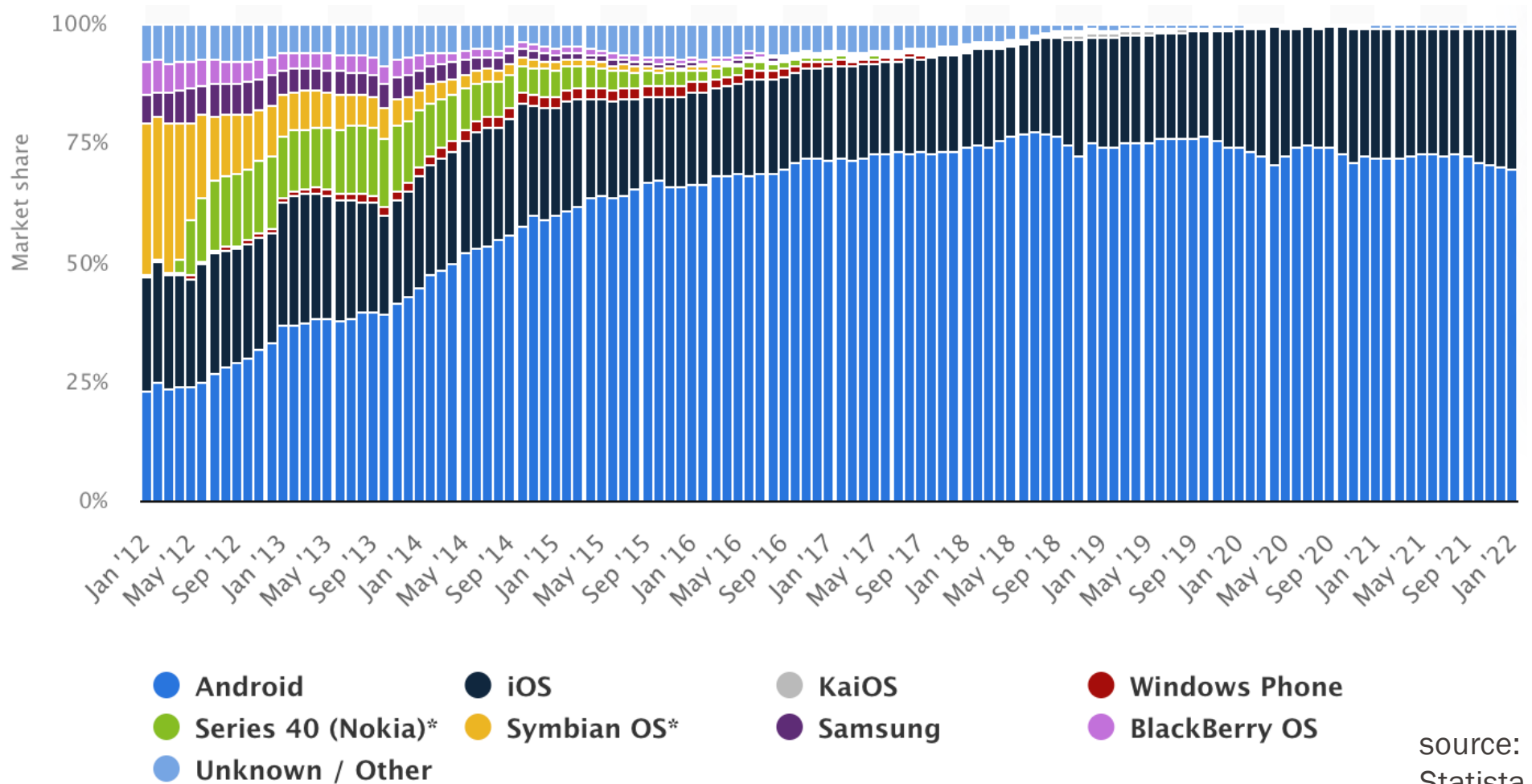
source: Statista, 2024

© ESL-EPFL



Source: GlobalVision, 2019

Android & iOS are the last two standing



source:
Statista, 2022

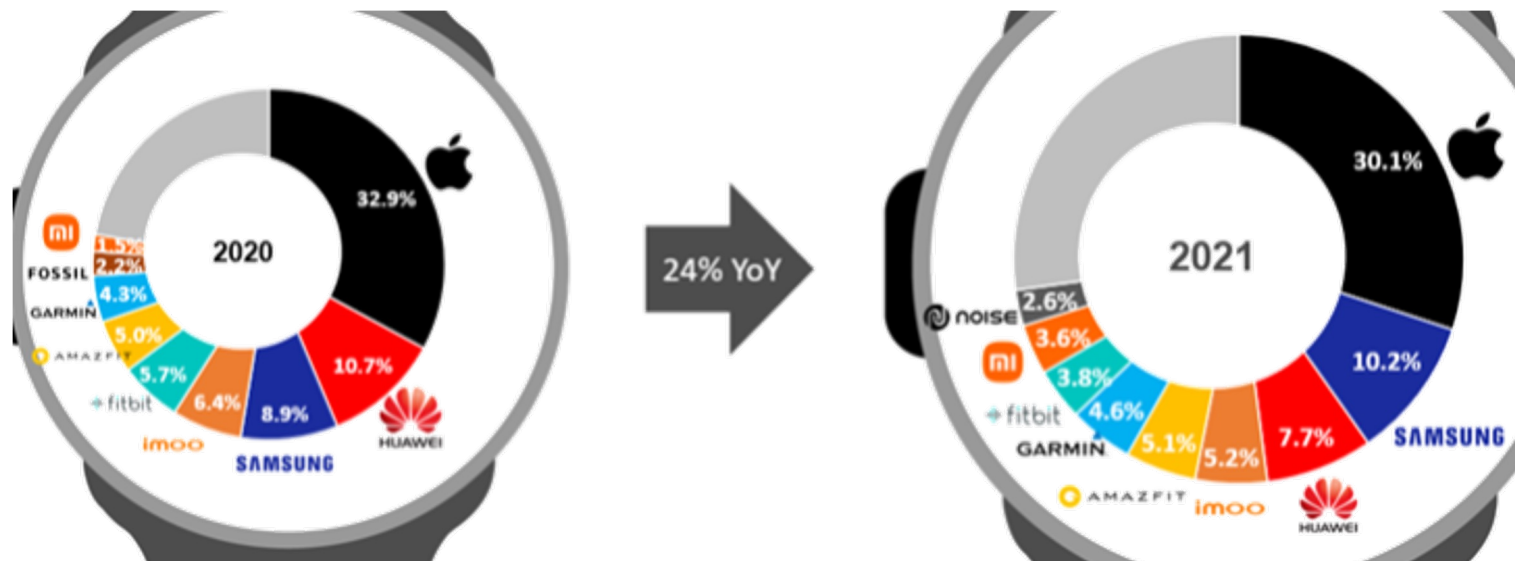
Expected **Market Size**
By 2023

**\$84.90
Billion**

Source: The Business
Research Company, 2022

Developers are conquering... the wrist

- The same old “Android vs. Apple” game
 - but with more players



Source: CounterPoint Research, 2022

Developers are conquering... the home



Source: Developer Megatrends

- Entertainment
- Security
- Speakers
- Lighting
- Thermostats
- ...

SMART HOME MARKET IN US 2022-2026



Market growth will **ACCELERATE** at a **CAGR** of

17.90%



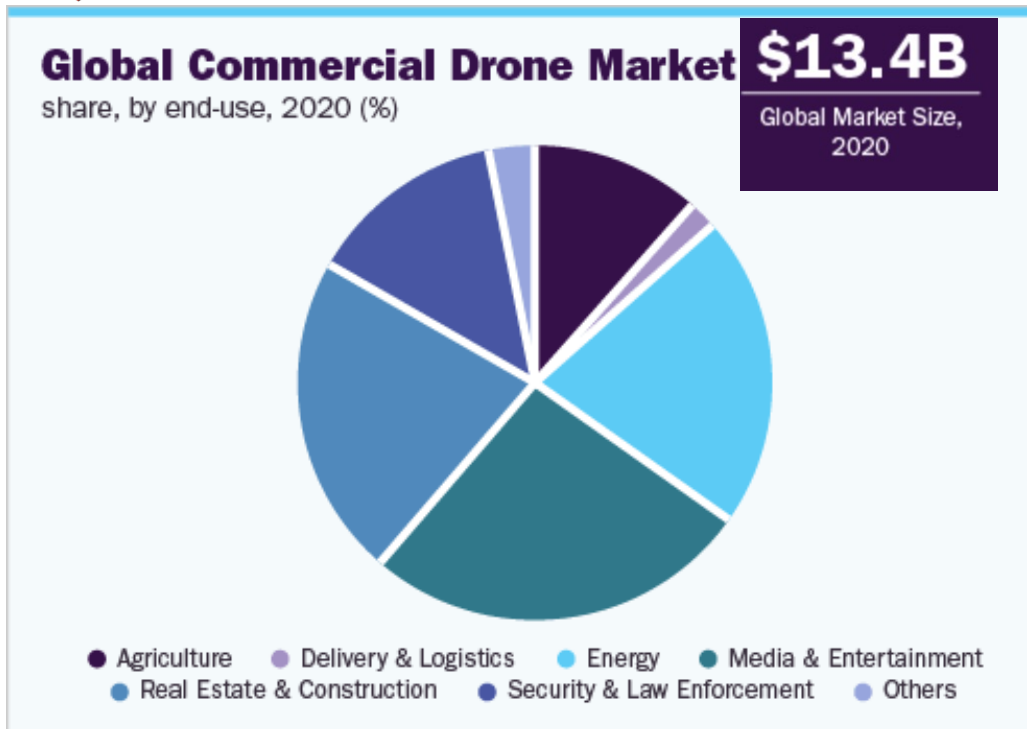
Incremental growth (\$B)

18.18

source: Technavio, 2022

Developers are conquering... the sky

■ Drones SDKs



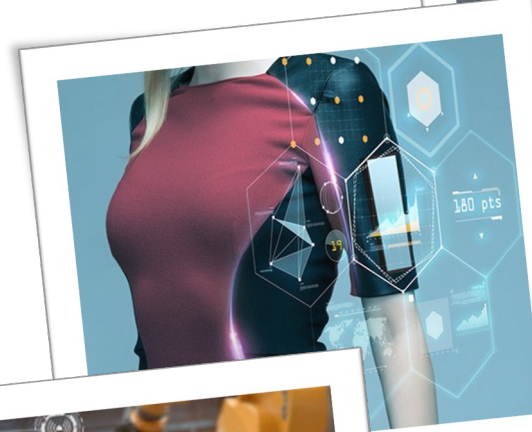
source: Grand View Research, 2020



Source:
Developer Megatrends

Developers are conquering...

- cars
- cities
- healthcare
- clothing
- factories
- ...





Android development Opportunities

- 17B IoT devices (2024), and growing
 - 32B devices in 2030 [1]
- Smart* are an increasing market
 - * = watches, drones, cities, cars, ...
- Endless application landscape
 - No dominant 3rd party developers.... yet
- **You can develop for them today!**

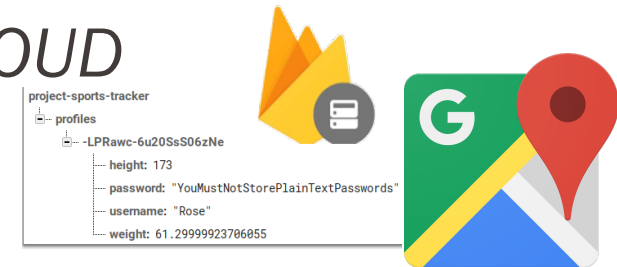




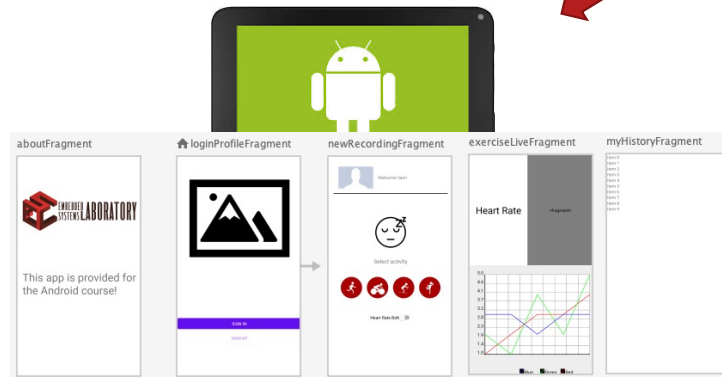
Distributed app across devices, sensors and the cloud



CLOUD



TABLET



WEARABLES



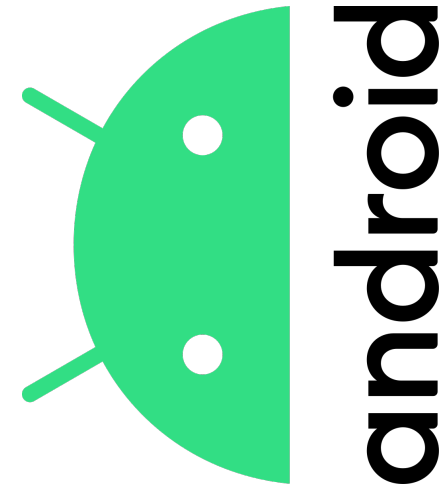


Class outline

- **Android overview**
- Resources, Composables
- Lab of Today
 - Create and build your first app for tablet & watch!

What is Android?

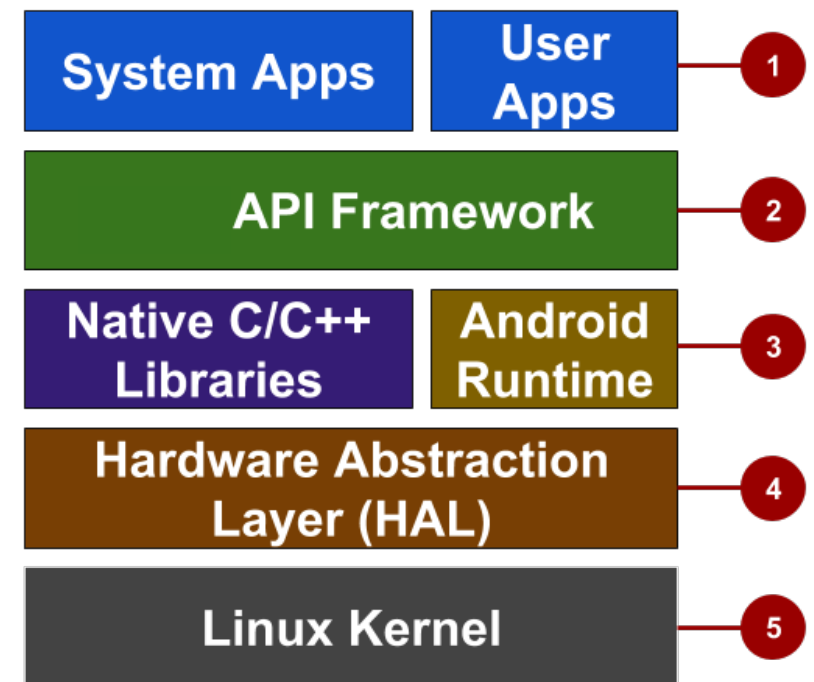
- Android is a **full ecosystem** used on **over 80%** of all smartphones
- Mobile operating system based on **Linux Kernel**
- User Interface (UI) for touch screens
- Multi-platform system:
 - powers devices such as watches, TVs, and cars
- Highly customizable for devices / by vendors
- Open source



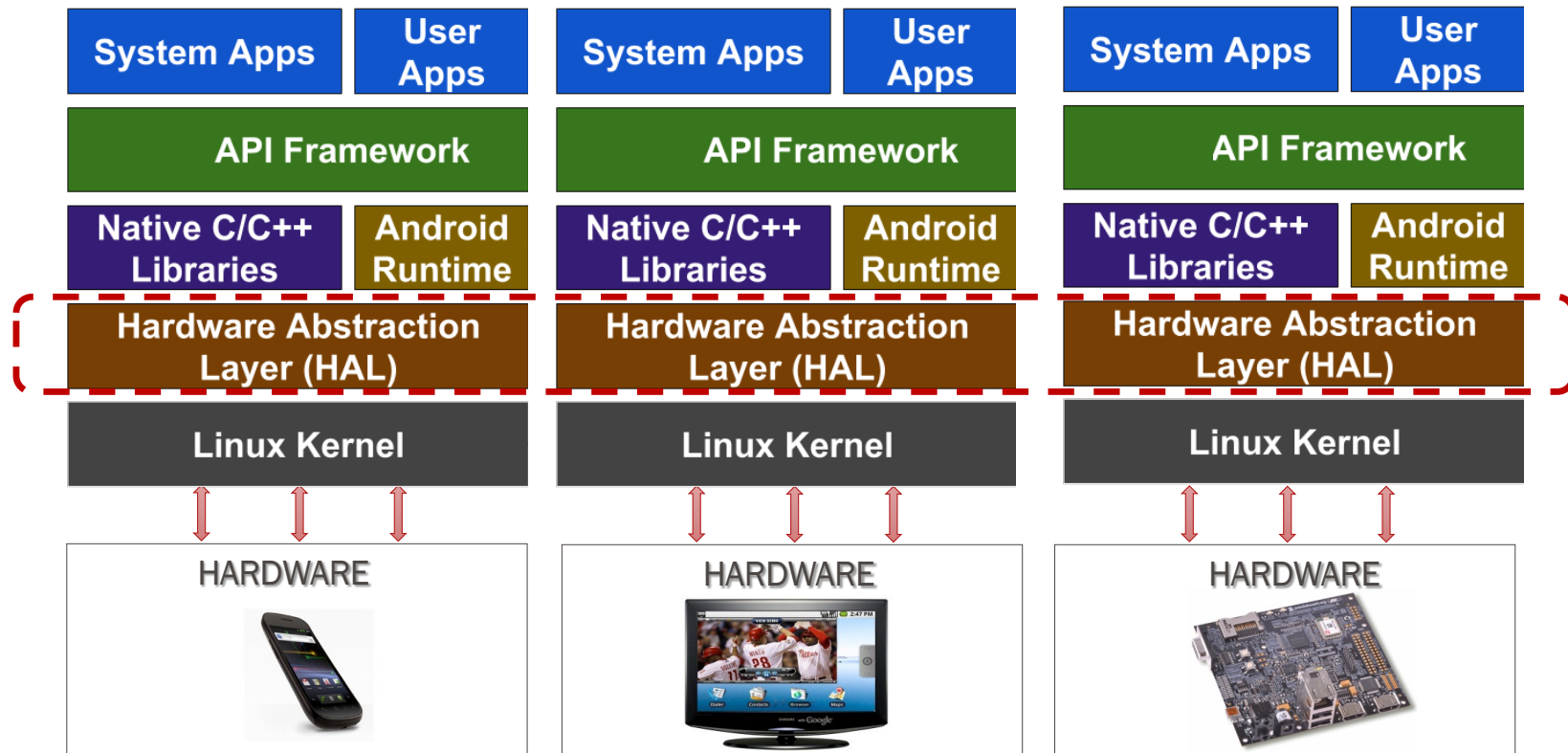
Android platform architecture

The Android stack is composed of layers:

1. System and user apps
2. Android OS API framework
 - Feature set of the Android-OS available via **Kotlin** APIs
3. Android runtime
 - Each app runs its own process with its own instance of the runtime
4. Hardware Abstraction Layer (HAL)
 - Standard interfaces that expose hardware as a library
5. Linux Kernel



Multi Platform Support



Huawei MediaPad T3 10"

- Chipset:
 - Qualcomm Snapdragon 425
 - 4 core Cortex A53 @1.4GHz
 - 3GB RAM
 - Built-in storage: 16GB
- Sensors
 - Ambient light sensor
 - Gyroscope
 - Accelerometer
 - Magnetometer
- Camera: 5MP (frontal 2MP)
- Bluetooth 4.0, A2DP
- WiFi 802.11 b/g/n
- 2G/3G/4G
- LCD Capacitive touchscreen
 - 9.6", 16million colors, 800x1280px, 157ppi



Width: 229.8 mm
About 460 g
(including the battery)

Height:
159.8 mm



Depth: 7.95 mm

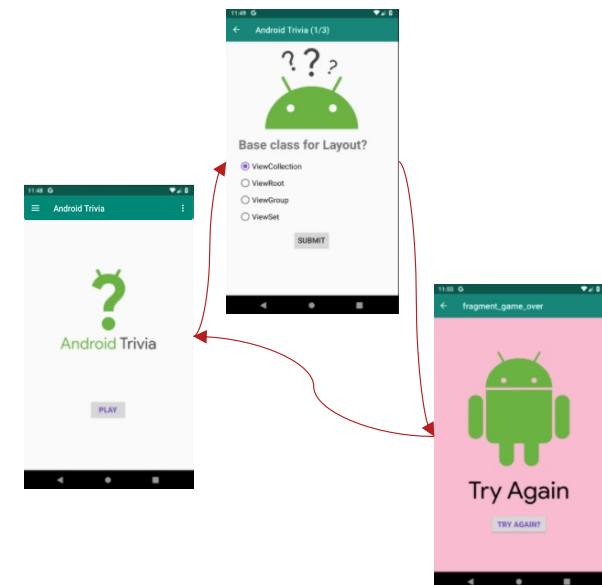
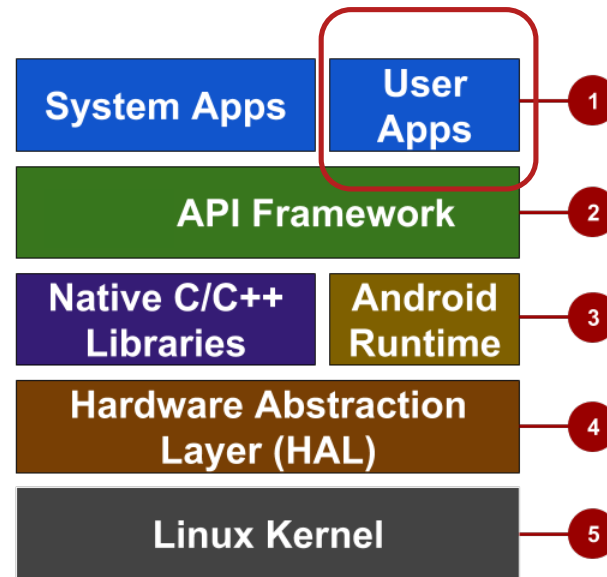
Huawei Watch 2

- Chipset:
 - CPU: Qualcomm Snapdragon 2100
 - 768MB RAM and 4GB Flash
- Sensors:
 - 6-axis a+G sensor
 - 3-axis compass
 - Heart Rate Sensor (PPG)
 - Barometer
 - Capacitive sensor
 - Ambient light sensor
- GPS
- WiFi 2.4GHz 802.11b/g/n
- Bluetooth 4.1
- Display:
 - 1.2-inch circular AMOLED display
 - 390x390 pixels w. 326 PPI



What is an Android app?

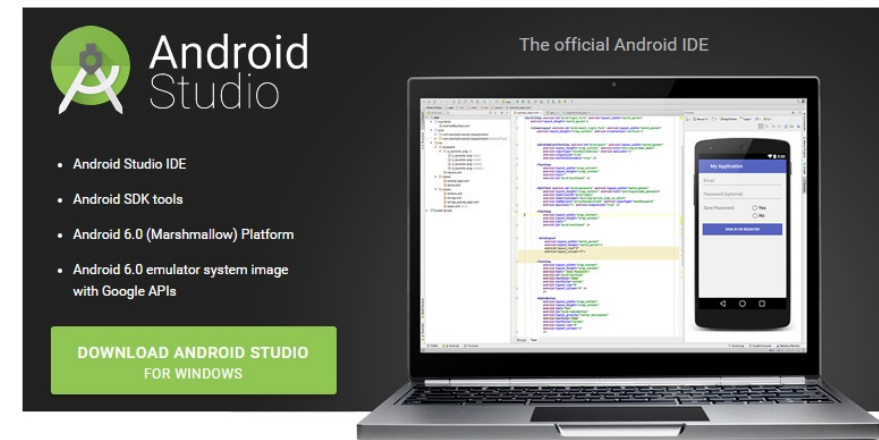
- A set of interactive screens
- GUI and behavior described in **Kotlin Programming Language**
- Developed with Android SDK (**Android Studio**)
- Using Android API Framework (2)
- Executed by Android Runtime Virtual machine (ART) (3)



Android Studio

- Developers can download Android Studio for free: <https://developer.android.com/studio/archive>
 - Development tools (debugger, monitors, editors)
 - Libraries (maps, wearables)
 - Virtual devices (emulators)
 - Documentation (<http://developers.android.com>)
 - Sample code

- Android itself is an Open Source Project: <http://source.android.com/>



Android versions



Source, Statcounter, July 2024

- Android 10 – 14 (2019-2024)
→ ~ 85% of devices
- App Developer specifies target and minimum version

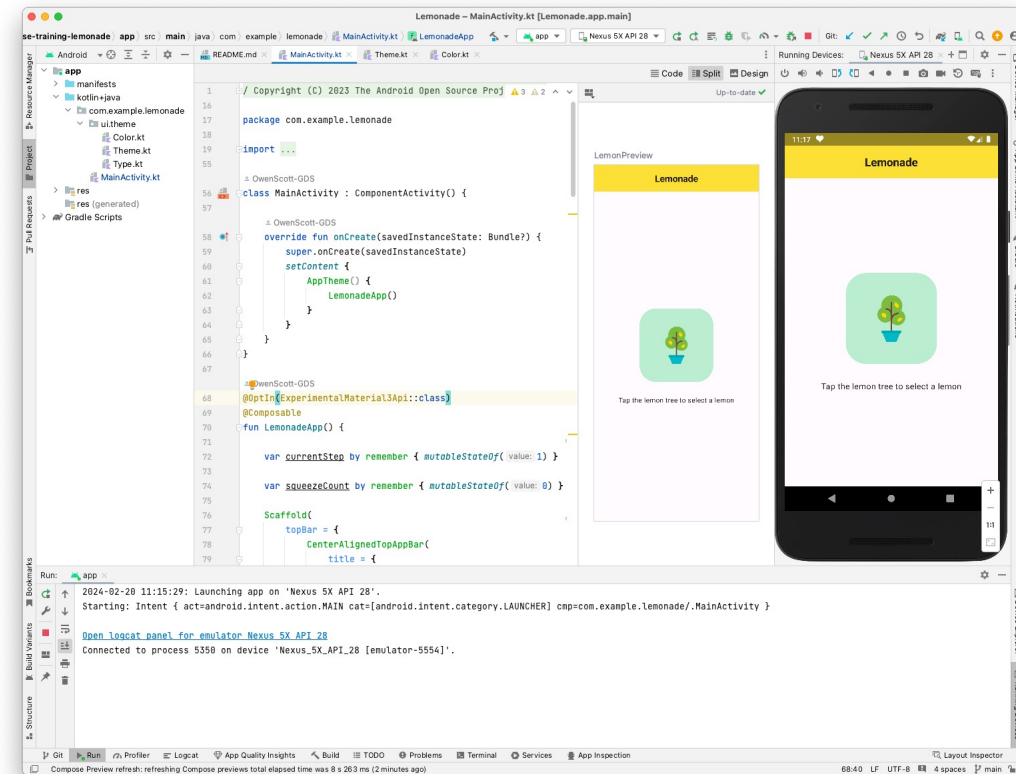
Android Studio

- Official Android IDE
 - Kotlin code for app logic and GUI layout
 - Configurations in XML

- Develop, run, debug apps ...
 - Visual layout preview
 - Virtual devices (emulators)

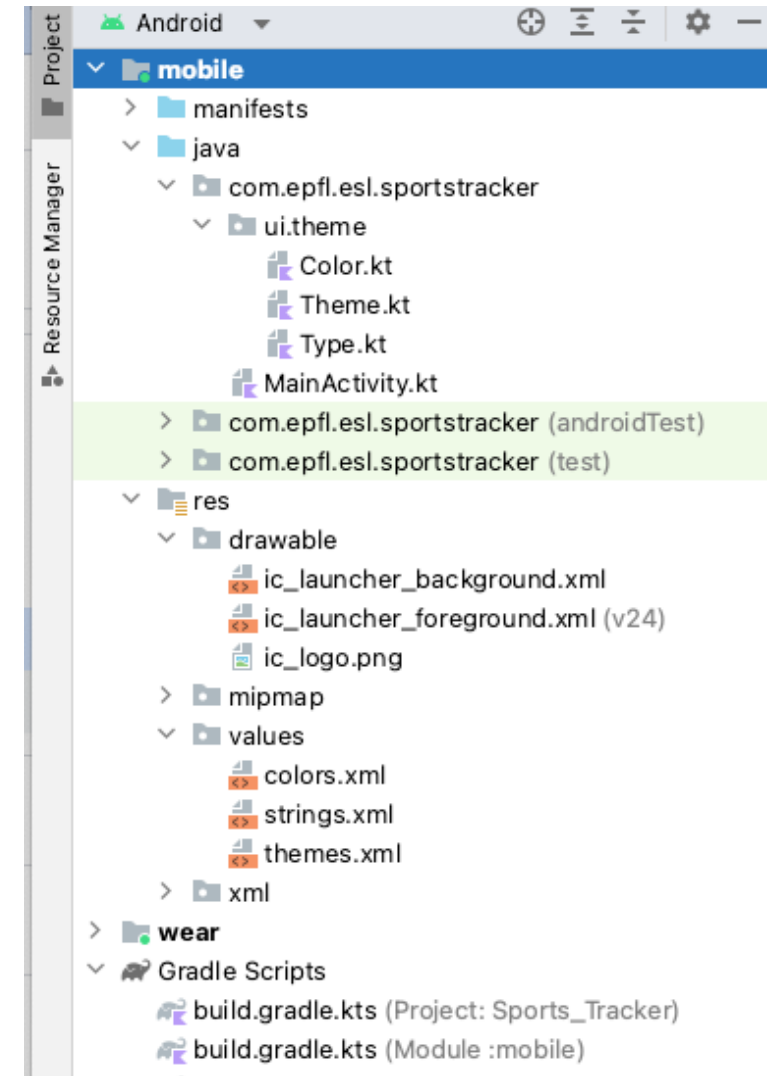
- Apps are packaged into APK or AAB files

- Apps published via the Google Play store
 - Official Google distribution service



Android application building blocks

- manifests:
 - Characteristics of the app and component definition
- java :
 - code of your app
 - Activities and VieModels → functionality
 - Composables → GUI
 - Helper classes
- res(ources):
 - images, strings, colors, XML and media files
- Gradle Scripts:
 - Scripts used to build the application





Class outline

- Android overview
- **Resources, Composables**
- Lab of Today
 - Create and build your first app for tablet & watch!

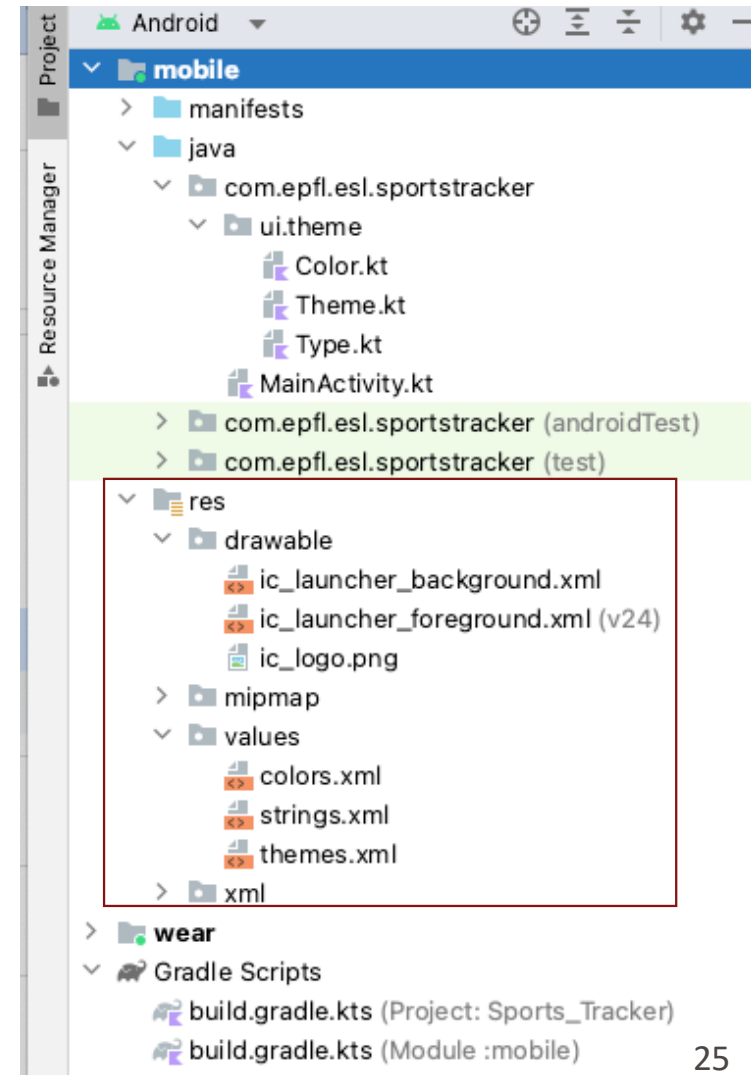


- Resources → everything that is not code

Resources... why?

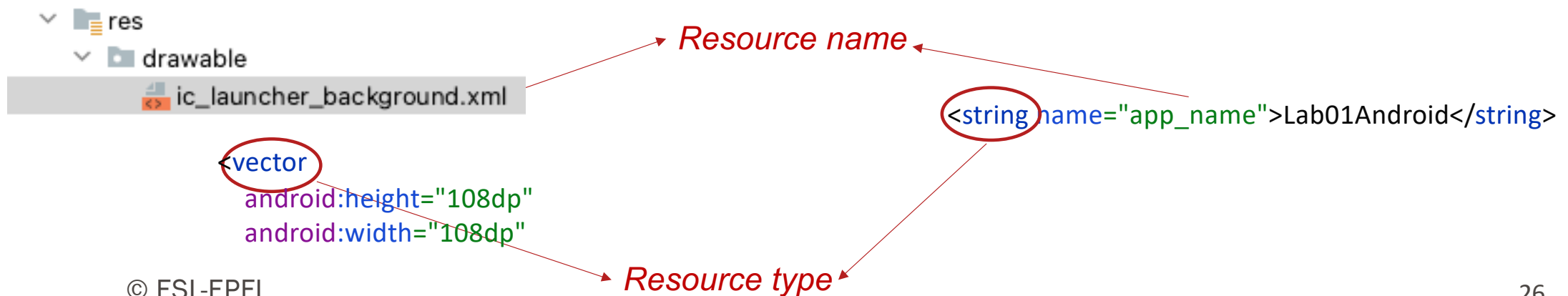
- Provide alternative resources to support specific device configurations
 - different language packs, different screen size images
 - re-compile only when needed

Application Resources



Application Resources

- Each Resource is associated with an **Identifier (ID)**
 - Identifiers must be unique!
- ID is composed of two parts:
 - The resource **type** (e.g. string, color, etc...)
 - The resource **name**, either:
 - the filename, excluding the extension (images/media)
 - a string in the XML `<name>` attribute (values)



Access to Application Resources in XML

- Referencing resources from Kotlin code

```
val myString = getString(R.string.app_name)
```

```
val myDdrawable =  
    getDrawable(R.drawable.ic_launcher_background)
```

Resource type

Resource name

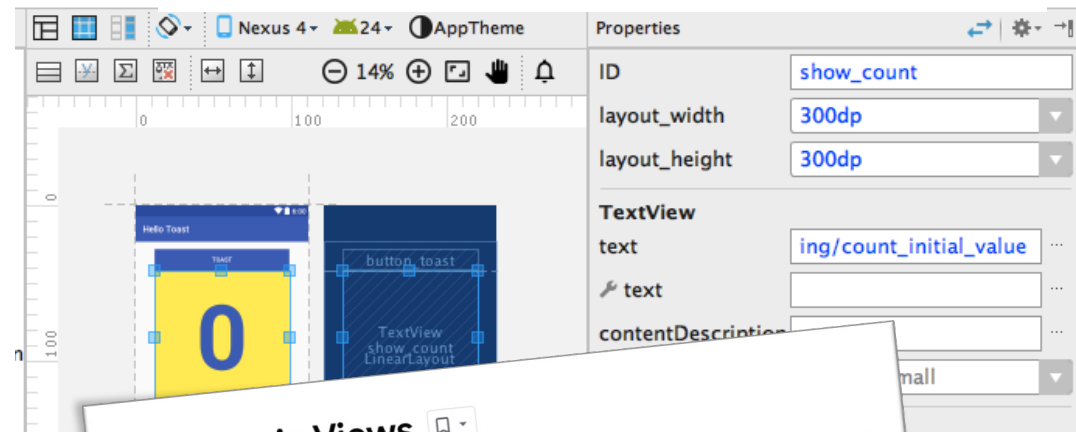
- R** class: glue between Kotlin and resources

- Using View resources
 - Text, buttons, ...
 - parameters for size, color, ...
- ViewGroups resources as "Views containers"
 - To organize GUI elements in columns, grids...
- Views-based GUI development
 - directly in XML code
 - graphically with Android Studio
- More information:

<https://developer.android.com/develop/ui/views/layout/declaring-layout>

GUI design in XML (the old way)

```
<TextView
    android:id="@+id/my_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="36sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



Layouts in Views

Try the Compose way

Jetpack Compose is the recommended UI toolkit for Android. Learn how to work with layouts in Compose.

[Learn Compose layout basics →](#)





- GUI elements declared as **Kotlin @Composable** functions

```
@Composable
fun Greeting(name: String) {
    Surface {
        Column(Modifier.padding(16.dp)) {
            Text(text = "Hello")
            Text(text = name)
        }
    }
}
```

Composables →

- Top composables called inside the **setContent** method in a ComponentActivity class
 - Activities are the “entry point” of an app (more on Activities next week)

- OnCreate() callback

- call to super
- inflate layout
- inflate theme
 - color scheme, typography, shapes
- MyApp() **composable** function

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                MyApp(modifier = Modifier.fillMaxSize())
            }
        }
    }
}

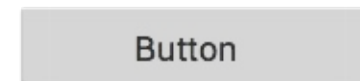
@Composable
fun MyApp() {
    ...
}
```



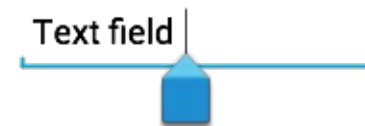
- **Composable** is the basic building block to create user interfaces
 - Everything the app user sees, is a composable

- Default/standard provided composables:
 - text display ([Text](#)), text edit ([TextField](#))
 - [Button](#)s, menus and other controls
 - scrollable lists: [LazyColumn](#), [LazyRow](#), [LazyGrid](#)
 - [Images](#)
 - ...

Button



EditText



SeekBar



Check
box



Radio
Button



Switch



- Composable can be `@Previewed`
 - previews can be parameterized to test specific scenarios or layout dimensions

```
@Composable
fun Greeting(name: String){
    Surface{
        Column(Modifier.padding(16.dp)) {
            Text(text = "Hello")
            Text(text = name)
        }
    }
}

@Preview(widthDp = 320, heightDp = 320)
@Composable
fun GreetingPreview() {
    MaterialTheme {
        Greeting("Android")
    }
}
```

Preview size

GreetingPreview

Hello
Android

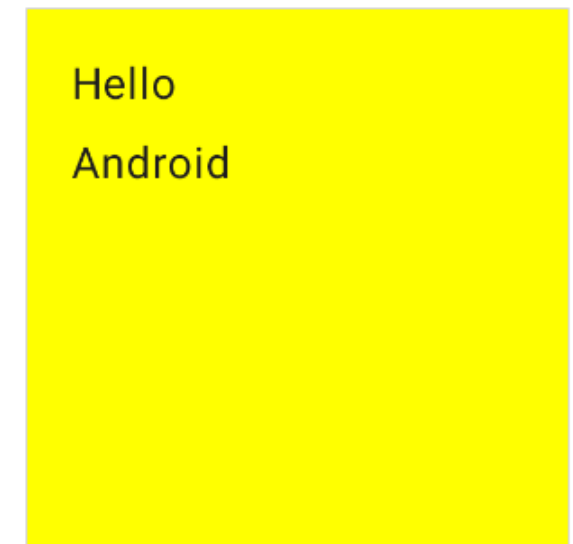


- Composables can be

- created
- reused
- combined

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier)
{
    Surface{
        Column(
            modifier = Modifier
                .background(color = Color.Yellow)
                .padding(16.dp)
        ) {
            Text(text = "Hello")
            Spacer(modifier = Modifier.size(8.dp))
            Text(text = name)
        }
    }
}
```

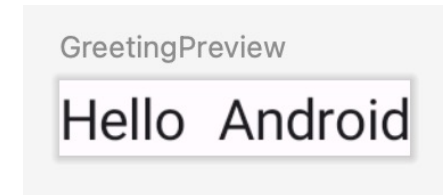
GreetingPreview



- Organize composables positions

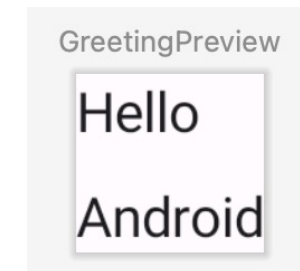
- Row

```
Row { this: RowScope
    Text(text = "Hello")
    Spacer(modifier = Modifier.size(8.dp))
    Text(text = name)
}
```



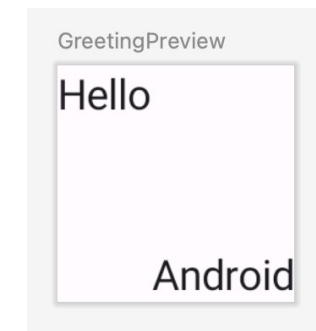
- Column

```
Column { this: ColumnScope
    Text(text = "Hello")
    Spacer(modifier = Modifier.size(8.dp))
    Text(text = name)
}
```



- Box – put elements on top of another

```
Box(modifier = Modifier.size(80.dp)) { this: BoxScope
    Text(text = "Hello", modifier = Modifier.align(Alignment.TopStart))
    Text(text = name, modifier = Modifier.align(Alignment.BottomEnd))
}
```





- Change composables size, layout, behavior and appearance
- Also, make composables clickable, scrollable, draggable or zoomable
 - next lectures!

```
Column(  
    modifier = Modifier  
        .background(color = Color.Yellow)  
        .padding(16.dp)  
        .fillMaxWidth()  
        .clickable { /* Handles click event */ }  
) { this: ColumnScope  
    Text(text = "Hello")  
    Spacer(modifier = Modifier.size(8.dp))  
    Text(text = name)  
}
```



GreetingPreview

Hello
Android



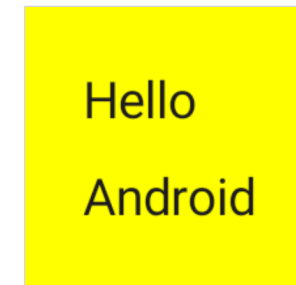
- The order of modifier functions is significant (left to right)

Example 1:

```
Column(  
  modifier = Modifier  
    .background(color = Color.Yellow)  
    .padding(16.dp)  
) { this: ColumnScope  
  Text(text = "Hello")  
  Spacer(modifier = Modifier.size(8.dp))  
  Text(text = name)  
}
```



GreetingPreview

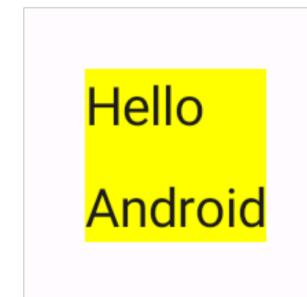


Example 2:

```
Column(  
  modifier = Modifier  
    .padding(16.dp)  
    .background(color = Color.Yellow)  
) { this: ColumnScope  
  Text(text = "Hello")  
  Spacer(modifier = Modifier.size(8.dp))  
  Text(text = name)  
}
```



GreetingPreview





- Some modifiers are specific for one type of composable scope
 - E.g. *Modifier.align(Alignment.BottomEnd)* can only be applied inside a **BoxScope**
 - Attempting to add a modifier to a wrong scope results in a compilation error

```
Box(modifier = Modifier.size(80.dp)) { this: BoxScope
    Text(text = "Hello", modifier = Modifier.align(Alignment.TopStart))
    Text(text = name, modifier = Modifier.align(Alignment.BottomEnd))
}
```

- Full list of modifiers and their scopes:
<https://developer.android.com/jetpack/compose/modifiers-list>

Passing modifiers between composables



- Best practice: add a modifier parameter at the end of custom Composables.
 - Makes reusing Composables easier and avoids code duplication

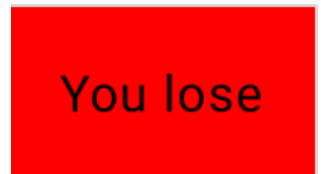
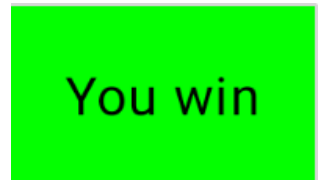
```
@Composable  
fun Message(message: String, modifier: Modifier = Modifier) { ... }
```

```
@Preview  
@Composable  
fun WinMessagePreview() {  
    MaterialTheme {  
        Message("You win", Modifier.background(Color.Green).padding(16.dp))  
    }  
}
```

Same
composable

```
@Preview  
@Composable  
fun LoseMessagePreview() {  
    MaterialTheme {  
        Message("You lose", Modifier.background(Color.Red).padding(16.dp))  
    }  
}
```

Different
modifiers





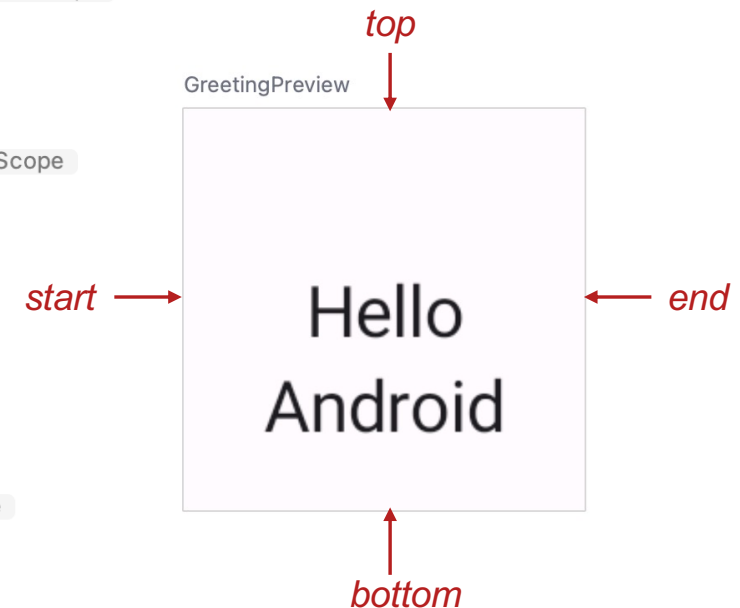
- Allows placement of composables relative to other composables on the screen
 - Avoids nesting multiple Rows and Column
 - Modifiers anchor composables to other composables

*reference
for each
composable
in the layout*

```
ConstraintLayout(modifier = Modifier.size(80.dp)) { this: ConstraintLayoutScope
    val (greetingRef, nameRef) = createRefs()

    Text(text = "Hello",
        modifier = Modifier.constrainAs(greetingRef) { this: ConstrainScope
            top.linkTo(parent.top)
            start.linkTo(parent.start)
            end.linkTo(parent.end)
            bottom.linkTo(parent.bottom)
        })

    Text(text = name,
        modifier = Modifier.constrainAs(nameRef) { this: ConstrainScope
            top.linkTo(greetingRef.bottom)
            start.linkTo(parent.start)
            end.linkTo(parent.end)
        })
}
```

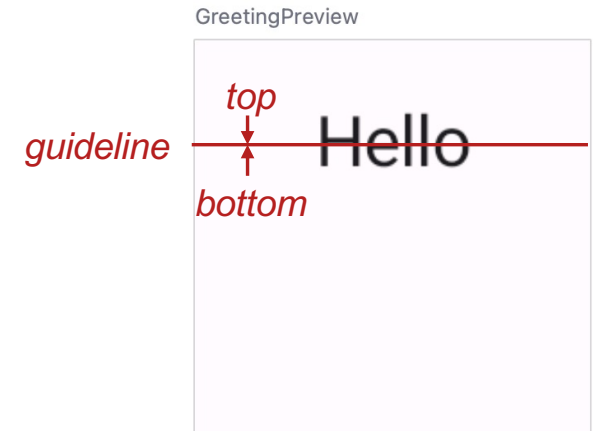




- Guidelines are invisibles lines that help to position composables
 - can be created at a certain point (dp) or certain percentage inside the parent composable.
- Two types:
 - Horizontal -> start and end
 - Vertical -> top and bottom

```
val topGuideline = createGuidelineFromTop(fraction: 0.25f)
```

```
Text(text = "Hello",  
    modifier = Modifier.constrainAs(greetingRef) { this: ConstrainScope  
        top.linkTo(topGuideline)  
        bottom.linkTo(topGuideline)  
        start.linkTo(parent.start)  
        end.linkTo(parent.end)  
    }  
)
```





- Recomposition is performed at run-time anytime the GUI needs to be updated
 - Drawing the screen is costly
- → Jetpack compose only redraws the GUI parts that needs updating
 - Composable functions can execute in any order depending on changes in internal state

```

@Composable
fun ButtonRow() {
    MyFancyNavigation {
        TopScreen()
        MiddleScreen()
        BottomScreen()
    }
}

```

← *TopScreen() does not have to execute before MiddleScreen()*



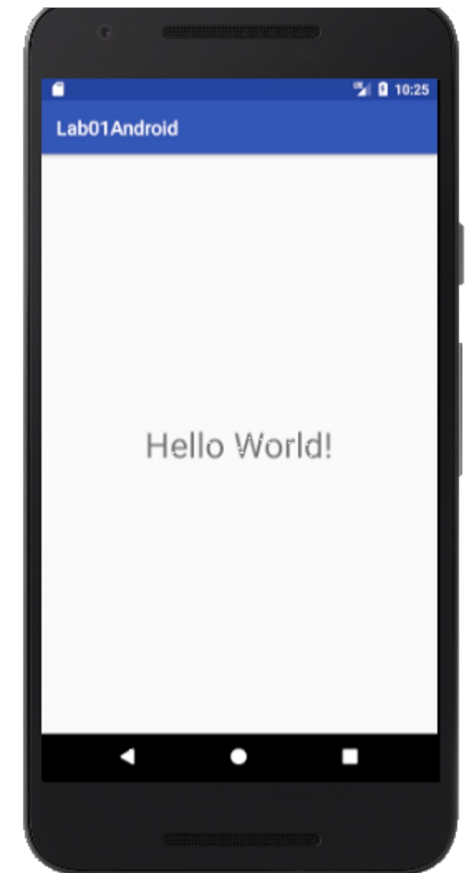
Class outline

- Android overview
- Resources, Composables
- **Lab of Today**
 - Create and build your first app for tablet & watch!



Today's Lab – Your first app

1. Creating your first Android project
2. A static screen displaying text and images
3. Use Column and ConstraintLayout
4. Running your app (on real device and emulator)
 1. on the Phone/Tablet
 2. on the Watch





Class outline

- Android overview
- Resources, Views and Layouts
- Lab of Today
 - Create and build your first app for tablet & watch!
- **Gentle reminder: groups and material**

Questions?

