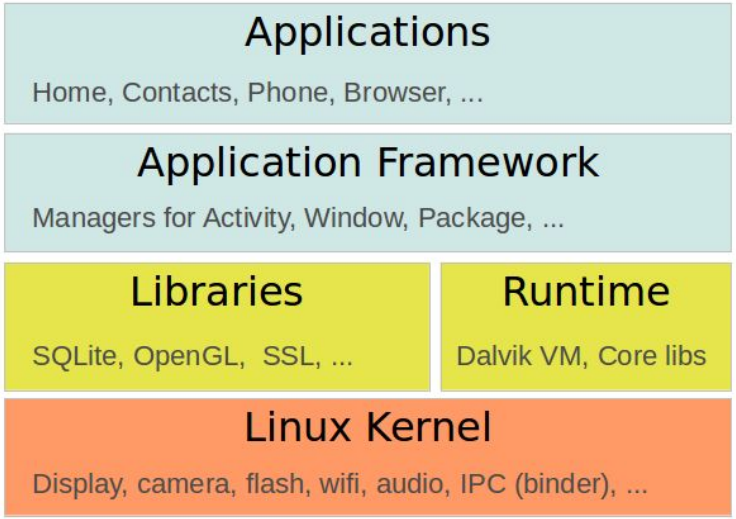
Sensor Based Mobile Applications
TX00CK66

Android recap, Application components, Development tools, Resources, Jetpack Compose UI, Activity - 22.08.2022

Patrick.Ausderau@metropolia.fi, Jarkko.Vuori@metropolia.fi

Android and its Application environment

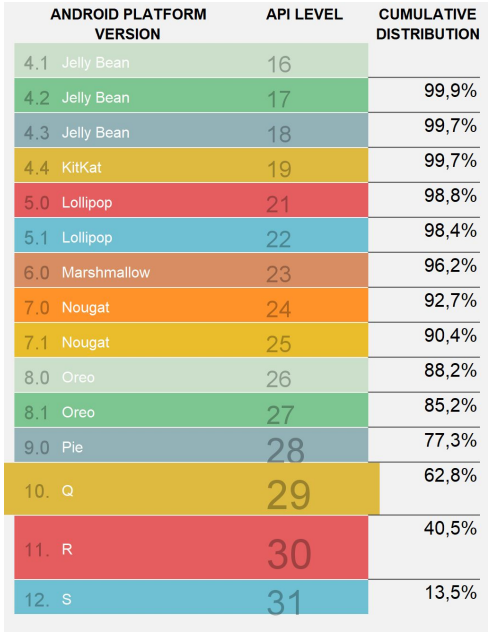
- Developed by a team led by Andy Rubin
  - Andy's nickname was Android because he was so interested about robots
  - Google acquired the team at 2005. Android version 1.0 was introduced on November 2007 (after the Apple iPhone was announced on January 2007), In September 2008, first Android phone
- Each application runs as separate Linux user and can by default access only its own files
  - Linux was chosen instead of proprietary OS in order to speed up development work
- For an application with a UI, the application integrates with the event processing loop in the framework - processes events the user triggers by his/her actions
- By default each application runs in its own process that has at minimum one thread - main thread where all UI events are processed



Dalvik VM was the original run-time interpreter (like Java VM), now replaced with Android Runtime, ART

New versions released frequently

- E.g. targeting Android >= 5.x makes your app running on more than 98% of the active devices on the play store
  - Basic concepts have not changed so much since 3.2
  - But some new features requires higher version
    - Material Design needs Android >= 5.x
    - ARCore needs Android >= 7.x
    - etc. https://developer.android.com/about/versions/
- Distribution dashboard, https://developer.android.com/about/dashboards/index.html, provides information about the relative number of devices that share a certain characteristic, such as screen size and density

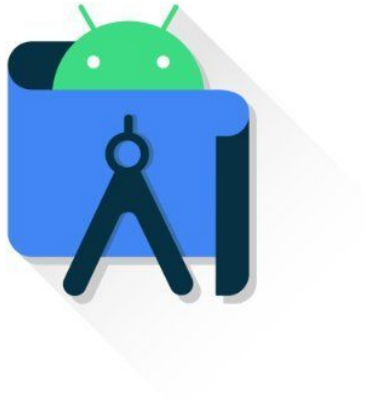


ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.1 Jelly Bean	16	
4.2 Jelly Bean	17	99,9%
4.3 Jelly Bean	18	99,7%
4.4 KitKat	19	99,7%
5.0 Lollipop	21	98,8%
5.1 Lollipop	22	98,4%
6.0 Marshmallow	23	96,2%
7.0 Nougat	24	92,7%
7.1 Nougat	25	90,4%
8.0 Oreo	26	88,2%
8.1 Oreo	27	85,2%
9.0 Pie	28	77,3%
10. Q	29	62,8%
11. R	30	40,5%
12. S	31	13,5%

You can find platform version information in Android Studio's Create New Project wizard

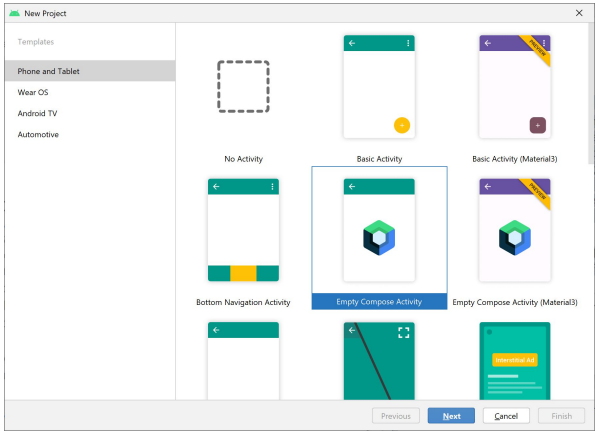
Tools

- Android Studio:
  - https://developer.android.com/studio/intro
  - Windows, Mac OS X, Linux
  - Check requirements: https://developer.android.com/studio/index.html#Requirements



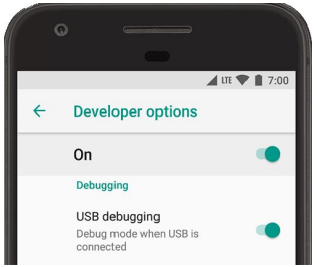
Lab 0 - Hello World

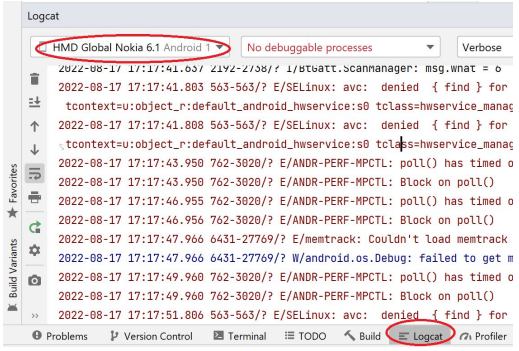
- Run Android Studio and create a new Project
  - Choose Empty Compose Activity
  - Application name, e.g. “Hello”
  - Choose a good package name, e.g. “fi.johndoe.hello”
  - Choose Kotlin language (no choice anyway!)
  - Choose Phone and Tablet with target at least API 23 (Android 6.0),
    - have a look at “Help me choose”
    - you may also use the API version you have on your phone (if you want later use your app in your phone)
  - Keep MainActivity or name it, e.g. “HelloActivity”
- Build and Run
  - By default, there should be a ready emulator to run your app.
    If not, create one from the Android Virtual Device (AVD) Manager:



Lab 0 - Connect Phone to Android Studio

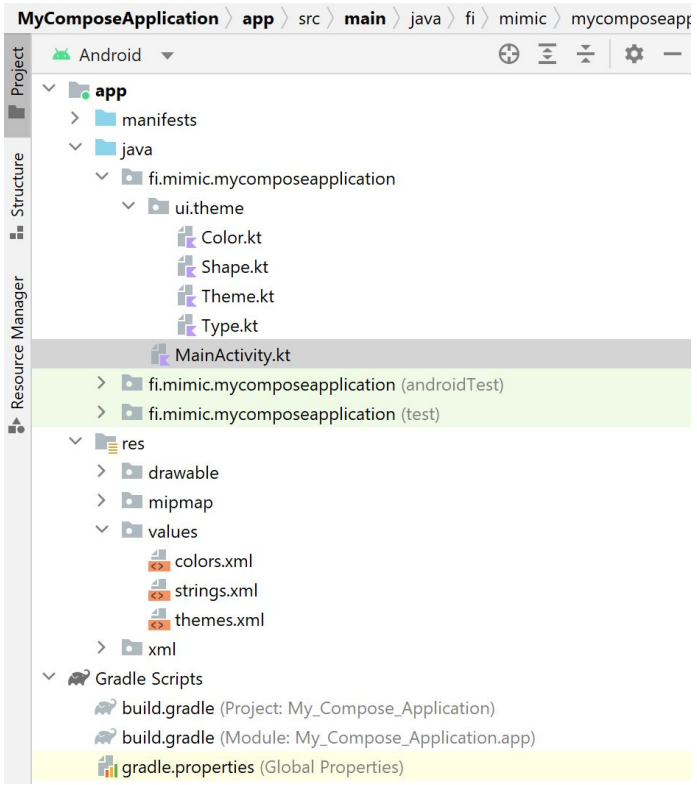
- Connect your phone/tablet to your PC using usb data cable. If your phone is Android ≥ 6, make sure it is in “USB for file transfer” mode (might work in charging mode too).
- Turn on “USB Debugging” on your device in Settings/Developer options.
  - To activate the Developer Options, check: https://developer.android.com/studio/debug/dev-options
- In Android Studio, click running devices to see if your device is attached
  - or with “Logcat”, https://developer.android.com/studio/debug/am-logcat
- If you have Android 11 phone or later, you are able to use WiFi instead of USB cable, see https://developer.android.com/studio/command-line/adb





Application Structure - Overview

- manifest
- source code
- resources
  - drawable
  - mipmap
  - values
- build.gradle

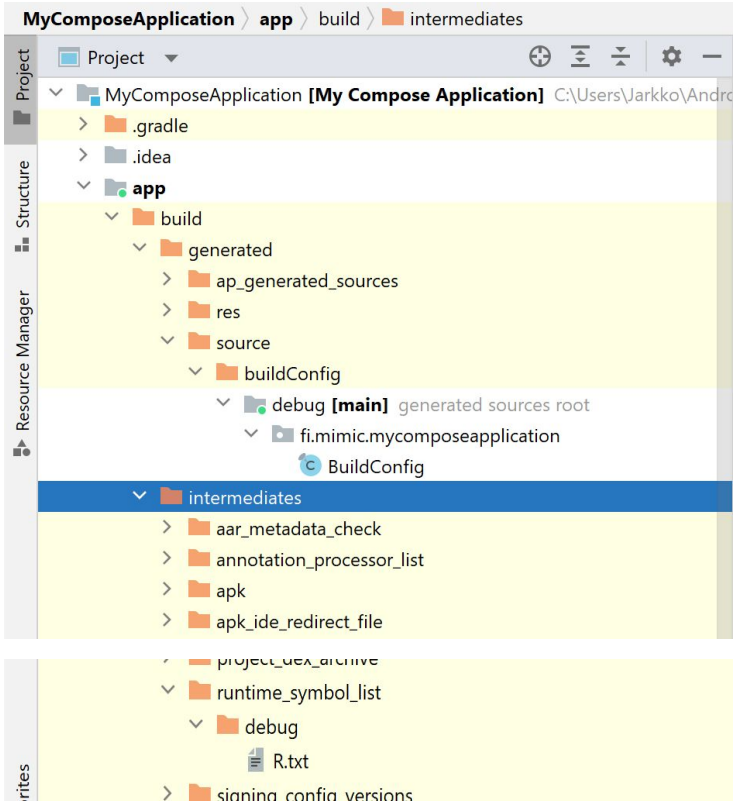


Generated files

- R
- BuildConfig

Never manually edit these files!
(Android Studio takes care of them)

They are provided for the Kotlin compiler to connect to the application environment (resources and build info)

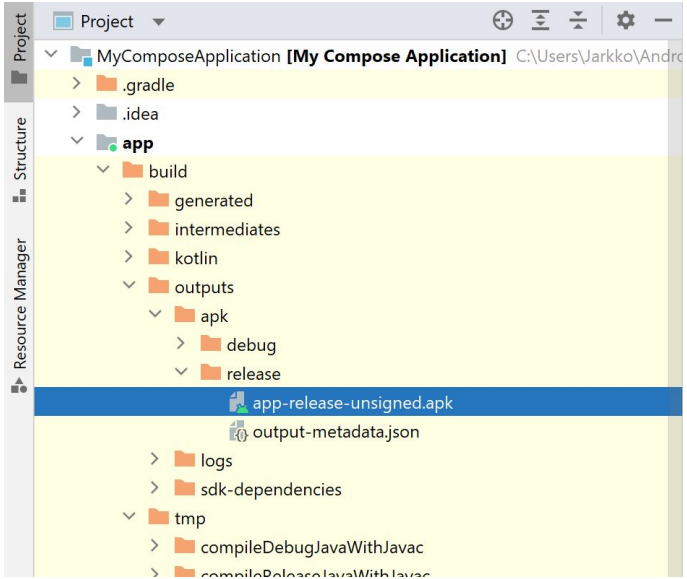


Gradle

- Android Studio uses Gradle (like sophisticated linux make), an advanced build toolkit to compile app resources and source code, and packages them into APKs that you can test, deploy, sign, and distribute
- At build time, Gradle generates the BuildConfig class so your app code can inspect information about the current build
  https://developer.android.com/studio/build
  https://developer.android.com/studio/build/gradle-tips

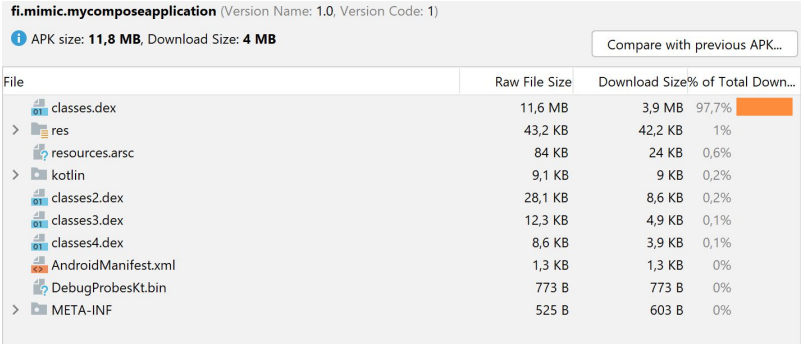
Build apk

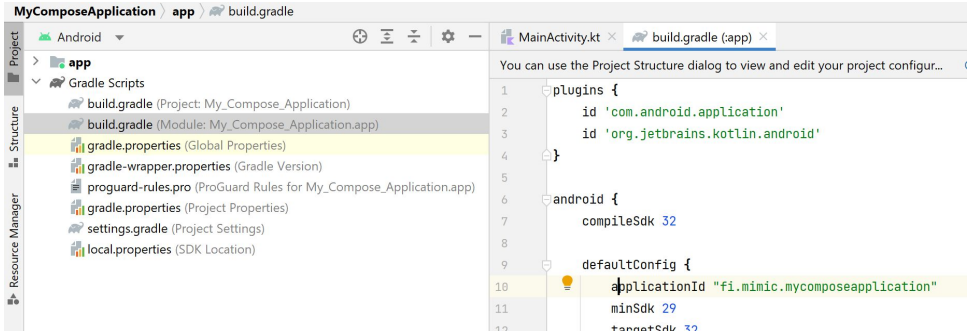
- Click Gradle on the right side of the IDE window
- Expand app (MyComposeApplication in this example) and double-click Task/other/assembleRelease
  - You may need to uncheck Gradle Settings/Experimental dialog ‘Do not build gradle task list during gradle sync’ settings (and then resync)
- From Run window you will see build status
- You will find .apk file in the AndroidStudioProjects (normally in your user home folder) from MyComposeApplication/app/build/outputs/apk - folder
- Android application is a zip-file, but with .apk extension



Application

- apk file (=Android application package)
  - Compiled classes (in Dalvik Executable format .dex)
  - Resources
  - Assets
  - Certificates
  - Manifest
- Each application
  - Runs in its own process
  - Runs in its own VM – isolation from other apps
  - Has its own unique (Linux) user ID – protection of app's files/db





Application Components

- A central feature of Android is that one application can make use of elements of other applications (provided those applications permit it)
  - For this to work, the system must be able to start an application process when any part of it is needed, and instantiate the Kotlin objects for that part
- Thus Android applications don’t have a single entry point for everything in the application - e.g. `main()` function
- There are four types of components:
  - Activities - represents a single screen with a user interface
  - Services - handles processing at the background, no UI
  - Broadcast Receivers - responds to system-wide broadcast announcements, no UI
  - Content Providers – access and management of other app’s data

Activities

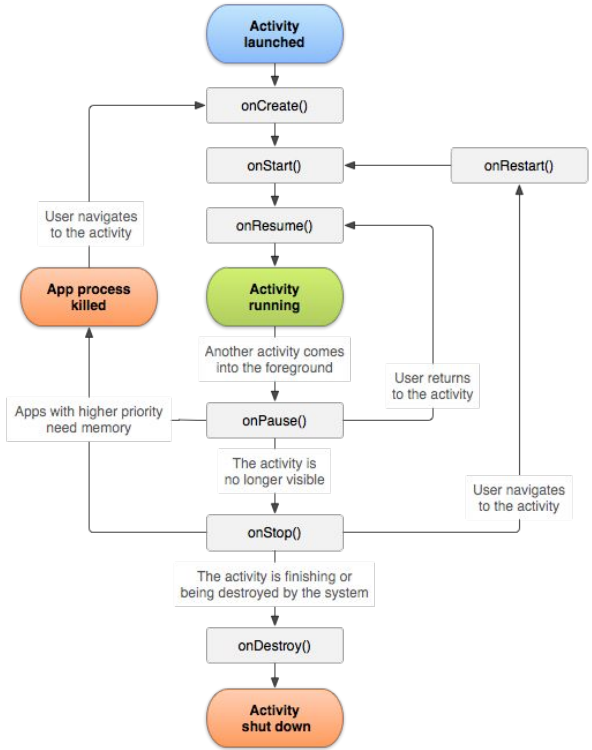
- Traditionally, an Activity represents a visual user interface for one functionality
  - Desktop systems have large screen areas to show large amounts of application functionalities simultaneously at the screen
  - Mobile phones usually have smaller screen size than desktop systems, therefore application functionalities must be divided to separate, smaller units (Activity)
- Example: a query application:
  - one activity displays a list of questions
  - second activity displays an answer form at a time
  - Third activity is reserved to change settings
  - Though activities work together to form a cohesive user interface, each activity is independent of the others
- Each activity is given a default window to draw in
  - Typically, the window fills the screen
- The visual content of the window is provided by a hierarchy of views — objects derived from the base View class
  - Each view controls a particular rectangular space within the window
  - Traditionally, this UI was described in XML in a separated resource layout file

Compose - `androidx.activity.ComponentActivity`

- There is a new way (other than XML layouts) to create UIs called Compose
- With Compose, an application typically use a single-activity architecture
- Each Views will be a `@Composable` element (function)
  - All the code is written in Kotlin (no more XML)
  - Less code
  - declarative API, “all you need to do is describe your UI - Compose takes care of the rest” https://developer.android.com/jetpack/compose/why-adopt
  - Better reusability
  - State is explicit and passed to the composable. Then, as app state changes, your UI automatically updates.
- Creating Compose application require an different way of thinking compare to the traditional way: https://developer.android.com/jetpack/compose/mental-model

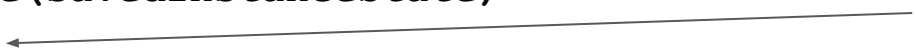
Activity Lifecycle

- onCreate() - called when the activity is first created
  - This is where you should do all of your normal static set up — create views, bind data to lists, instantiate your custom objects, and so on
  - This method is passed a Bundle object containing the activity's previous state, if that state was captured
- Note: any activity lifecycle method should always first call the superclass constructor
  - In order Android OS to initialize necessary data structures



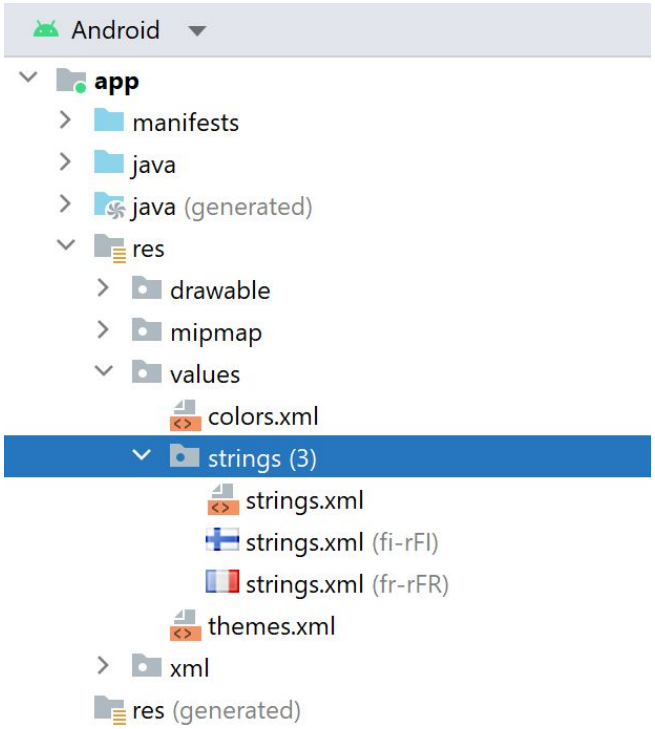
Activity Source Code

```
package fi.metropolia.mycomposeapplication

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.res.stringResource

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(stringResource(R.string.hello_world))
        }
    }
}
```



Resources

- Externalize resources (images, strings, etc.) from your application code, so that you can maintain them independently
- Alternative resources to support specific device configurations and localization, e.g.
  - values
  - values-fi
  - values-fr
- Note: default folder should include all needed resources



Resources - values

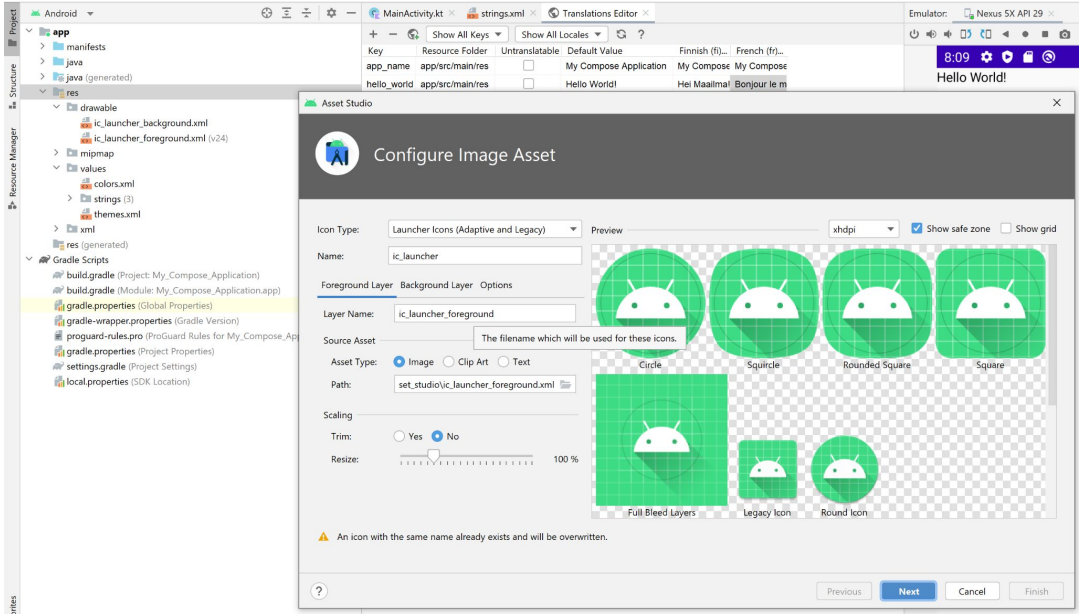
- XML files that can be compiled into many kinds of resource
  - arrays.xml for resource arrays (typed arrays)
  - colors.xml to define color drawables and color string values
    - Can be done in Compose too
  - dimens.xml to define dimension value
    - Can be done in Compose too
  - strings.xml to define individual string values
  - styles.xml to define the format and look for a UI (view, activity or application level)
    - Can be done in Compose too
  - …

Resources - drawable

- A drawable resource is a general concept for a graphic that can be drawn to the screen and which you can retrieve from Kotlin code or use in another XML resource
- .png, .jpg files that are compiled into bitmap files or 9-patches (resizable bitmaps)
  - mipmap - multiple resolution bitmap to render nicely to displays of different resolution
- Drawables can be used as launcher icons, menu item icons, (animated) background images in views, custom buttons, etc.

Resources - Mipmap

- Drawable files for different launcher icon densities.
- Using Asset Studio you can define easily your own launcher icon
  - Right-click the res folder and select New > Image Asset



Resources - values - strings.xml

.../app/src/main/res/values/strings.xml

```
<resources>
   <string name="app_name" translatable="false">My Compose Application</string>
   <string name="hello_world">Hello World!</string>
   <string name="hello_person">Hi %s!</string>
</resources>
```

.../app/src/main/res/values-fi/strings.xml

```
<resources>
   <string name="hello_world">Hei Maailma!</string>
   <string name="hello_person">Hei %s!</string>
</resources>
```

Resources - others…

- res/raw - The files will be traded as raw files, they will not be compiled
  - For example audio / video files that will be used by media players or static text files that can be read at run time
- res/xml - Arbitrary XML files that are compiled and can be read at run time
- …

Compose - Layout

- `Column()` - place items vertically on the screen

```
@Composable
fun UserInfo(user: User){
    Column {
        Text(user.name)
        Text(user.role)
    }
}
```

- `Row()` - place items horizontally on the screen
- `Box()` - put elements on top of another
- `Scaffold()` - provides slots for a top app bar or a bottom app bar (and more)
- `Surface()` - provides elevation (shadow)
- `Card()` - contains content and actions about a single subject
- See https://developer.android.com/jetpack/compose/layouts

Compose - `androidx.compose.material`

- `Text()` - display text
  https://developer.android.com/jetpack/compose/text

```
@Composable
fun Greetings(text: String) = Text(
    stringResource(R.string.hello_person, text)
)
```

- `Button()` - contains actions (that must be set as lambda function)

```
Button(onClick = { /* lambda callback function code */ }) {
    Text(stringResource(R.string.click)
}
```

- See also
  https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary
  and for Material Design
  https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary

Compose - `androidx.compose.material`

- `TextField()` - user inputs. Example with state management
  https://developer.android.com/jetpack/compose/state

```
@Composable
fun ModifyText() {
    var text by remember { mutableStateOf("") }
    Column {
        Greetings(text)
        TextField(
            value = text,
            onValueChange = { text = it },
            label = { Text(stringResource(R.string.enter_name)) }
        )
    }
}
```

- etc.
  - Checkbox, RadioButton, Slider, Switch,…

A Simple Android Application - Compose way

manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="fi.metropolia.mycomposeapplication">

   <application
       android:allowBackup="true"
       android:dataExtractionRules="@xml/data_extraction_rules"
       android:fullBackupContent="@xml/backup_rules"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.MyComposeApplication"
       tools:targetApi="31">
       <activity
           android:name=".MainActivity"
           android:exported="true"
           android:label="@string/app_name"
           android:theme="@style/Theme.MyComposeApplication">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
   </application>
</manifest>
```

string.xml

```
<resources>
    <string name="app_name" translatable="false">My Compose Application</string>
    <string name="hello_world">Hello World!</string>
    <string name="hello person">Hi %s!</string>
    <string name="click">Click me</string>
    <string name="enter_name" translatable="false">Enter Name</string>
</resources>
```

MainActivity.kt

```
package fi.metropolia.mycomposeapplication

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.*
import androidx.compose.ui.res.stringResource

class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContent {
           Scaffold(topBar = { TopAppBar(title = { Text(stringResource(R.string.app_name)) }) }) {
               Text(stringResource(R.string.hello_world))
       }
   }
}
```

Compose - Style

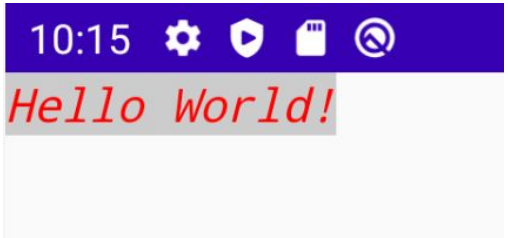
- Text as other elements can be styled
  https://developer.android.com/reference/kotlin/androidx/compose/ui/text/style/package-summary

```
Text(
    stringResource(R.string.hello_world).repeat(10),
    color = Color.Magenta,
    fontSize = 32.sp,
    fontWeight = FontWeight.Bold,
    letterSpacing = 2.sp,
    maxLines = 2,
    overflow = TextOverflow.Ellipsis,
    textDecoration = TextDecoration.Underline
)
```

Compose - Style

- For better code reuse, as for any `@Composable`, the style can be set separately

```
Text(stringResource(R.string.hello_world), style = MyTextStyle())

// define somewhere else, e.g. in ui package
@Composable
fun MyTextStyle() = TextStyle(
    color = Color.Red,
    fontSize = 16.sp,
    fontFamily = FontFamily.Monospace,
    fontStyle = FontStyle.Italic,
    background = Color.LightGray
)
```



- Example with TextStyle: https://developer.android.com/reference/kotlin/androidx/compose/ui/text/TextStyle

Compose - Modifier

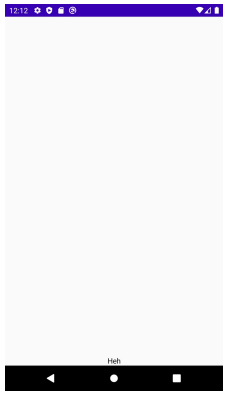
- Modifier - modifier elements that decorate or add behavior to Compose UI elements (see https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier)
  - Order matters!
    Modifier.padding().border().background()
    will not give the same result as
    Modifier.background().padding().border()


```
Text(
    stringResource(R.string.hello_world),
    textAlign = TextAlign.Center,
    modifier = Modifier
    .fillMaxWidth()
    .padding(5.dp) // same as css margin = outside spacing
    .border(BorderStroke(2.dp, Color.Blue))
    .background(Color.Green)
    .padding(8.dp) // same as css padding = inner spacing)
```

Compose - Alignment and Arrangement

- Alignment - used to define the alignment of a layout inside a parent layout https://developer.android.com/reference/kotlin/androidx/compose/ui/Alignment
- Arrangement - used to specify the arrangement of the layout's children in layouts like Row or Column in the main axis direction (horizontal and vertical, respectively) https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/Arrangement

```
// note the composable callback method parameter
// call like BottomCenterColumn { Text("Heh") }
@Composable
fun BottomCenterColumn(children: @Composable () -> Unit) =
   Column(
       horizontalAlignment = Alignment.CenterHorizontally,
       verticalArrangement = Arrangement.Bottom,
       modifier = Modifier
           .fillMaxHeight()
           .fillMaxWidth()
   ) { children() }
```



Referring resources from code

- Activity class takes care of creating a window for you in which you can place your UI
- `onCreate()` is where you initialize your activity - typically you call `setContent{}` with `@Composable` defining your UI
- Composable can refer to any resources
  - for text in strings.xml, e.g.
    `<string name="hello">Hello World!</string>`
    can be refer in the code with
    `stringResource(R.string.hello)`
  - same for other resources, e.g. colors.xml can be refer with
    `colorResource(R.color.black)` , etc.

UI Events Lambdas

- An event listener is an interface that contains a single callback method
  - These methods will be called by the Android framework when the Component to which the listener has been registered is triggered by user interaction with the item in the UI
- Some Composable have their own event as Lambdas (extended concept from anonymous class in Java), e.g.
  - `Button(onClick = { someFun() }) { Text("Click") }`
  - `TextField(onValueChange = { /* ... */ })`
- For other Composable it is possible to use, e.g.

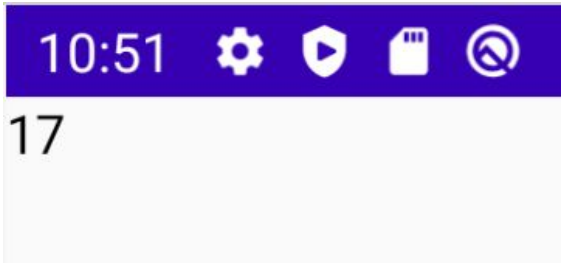
```
Modifier.clickable { /* ... */ }
```

Compose - State

- State in an app is any value that can change over time
  https://developer.android.com/jetpack/compose/state
  https://developer.android.com/reference/kotlin/androidx/compose/runtime/package-summary
- Composable functions can store a single object in memory by using the remember composable
  - A value computed by remember is stored in the Composition during initial composition, and the stored value is returned during recomposition
- mutableStateOf creates an observable MutableState<T>, which is an observable type integrated with the compose runtime

Compose - State example

```
// preview makes possible to see how the ui looks like in Android Studio
// without having to run the app
@Preview
@Composable
fun MainView() {
    val count = remember { mutableStateOf(0) }
    Column(
        modifier = Modifier
            .fillMaxHeight(fraction = 0.5f)
            .fillMaxWidth()
            .clickable { count.value++ }
    ) {
        Text(count.value.toString())
    }
}
```



Reading list

- Kotlin
  - Jemerov, Isakova: “Kotlin in Action”, Manning, 2017