University of the West of England
BRISTOL

Week 2: First Android App

UFCF7H-15-3 Mobile Applications

Dr Kun Wei

Introduction to Android Studio

- Official IDE for Android app development.
- Besides user-friendly interface and code editor, it includes layout editor (the view-based layout is not recommended).
- Gradle Build System for managing libraries and resources efficiently.
- **Emulator** and Device Support
- Debugger and Version Control Integration
- Instant Run (but takes quite a while to compile)
- Support for Libraries and APIs
- Google Play Integration

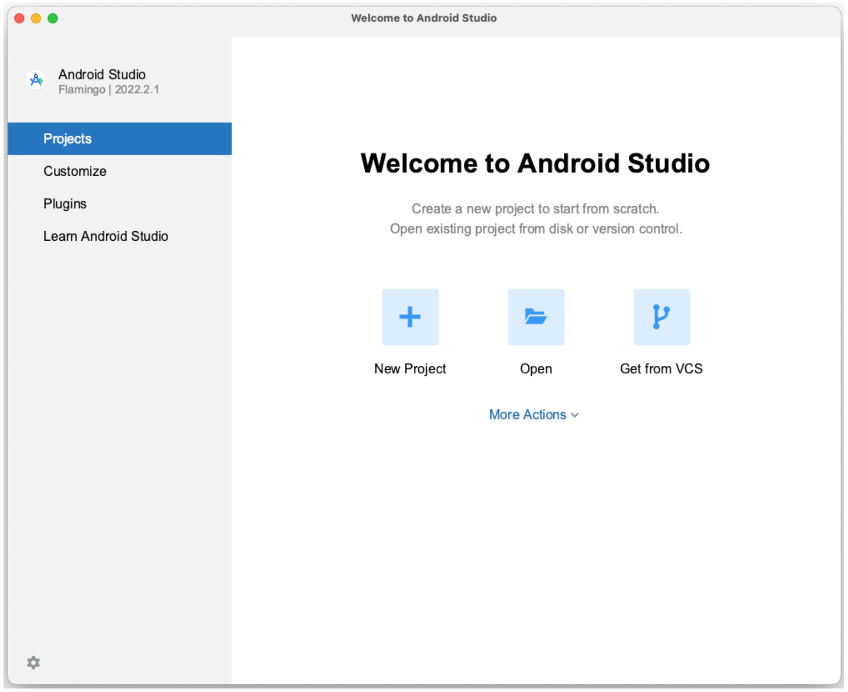
University of the West of England BRISTOL

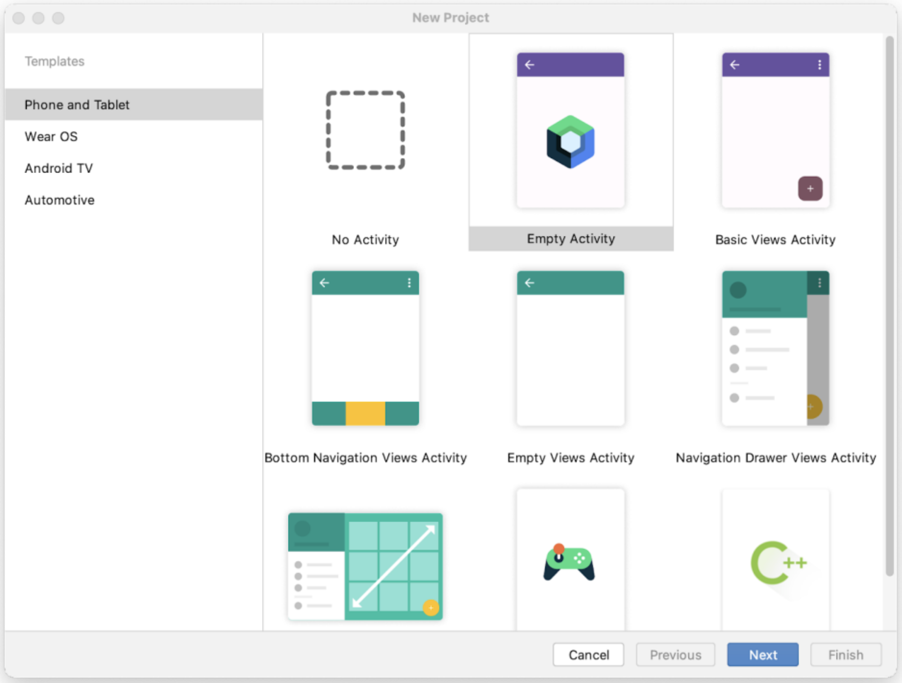
Kotlin and Jetpack Compose

- Kotlin is a modern statically typed programming language used by over 60% of professional Android developers that helps boost productivity, developer satisfaction, and code safety.

- Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.

Create an empty activity project

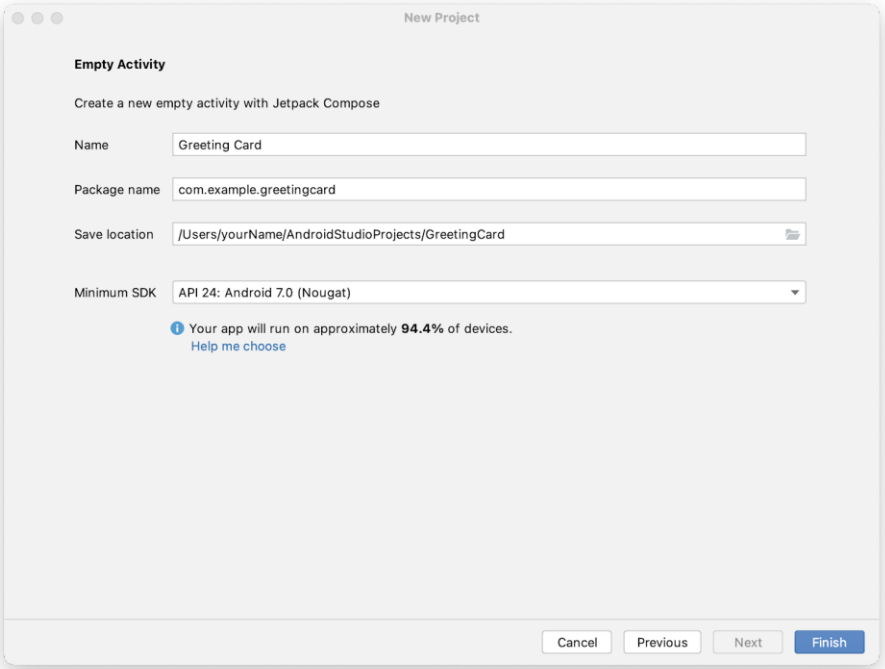

Welcome to Android Studio
Android Studio
Flamingo | 2022.2.1
Projects
Customize
Plugins
Learn Android Studio
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions



Usually, AS takes **quite a while** to create a new project.

University of the West of England
BRISTOL

Create your first app



New Project

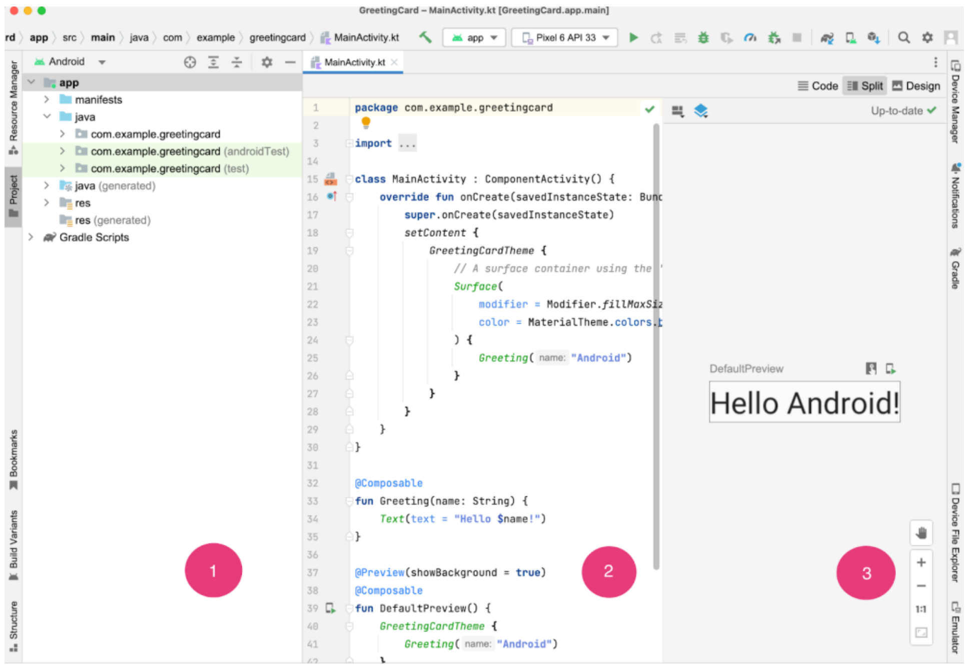
Empty Activity

Create a new empty activity with Jetpack Compose

Name	Greeting Card
Package name	com.example.greetingcard
Save location	/Users/yourName/AndroidStudioProjects/GreetingCard
Minimum SDK	API 24: Android 7.0 (Nougat)

Your app will run on approximately **94.4%** of devices.
Help me choose

Usually, AS takes **quite a while** to create a new project.

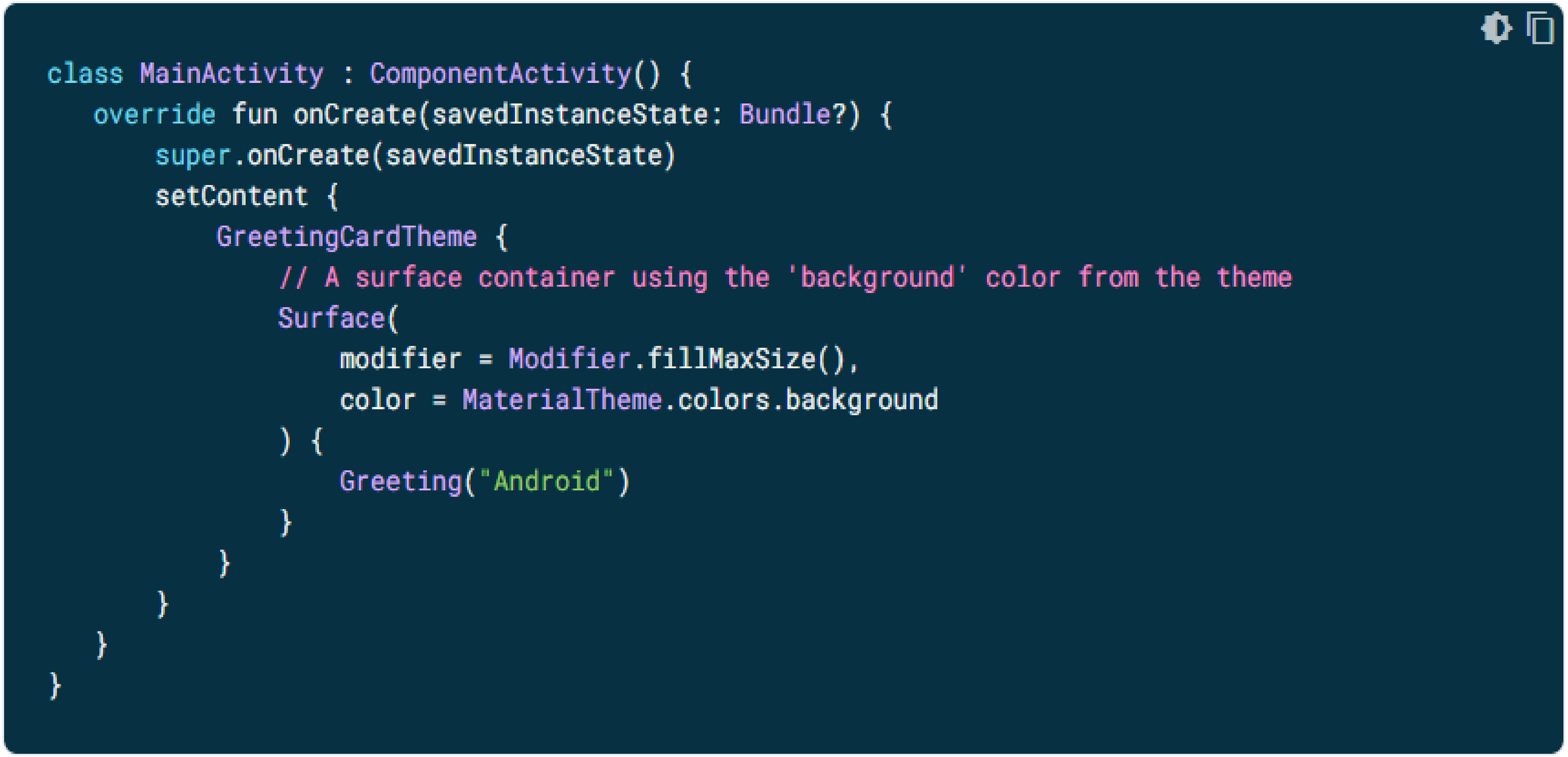


GreetingCard – MainActivity.kt [GreetingCard.app.main]

```
package com.example.greetingcard

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bunc
        super.onCreate(savedInstanceState)
        setContent {
            GreetingCardTheme {
                // A surface container using the '
                Surface(
                    modifier = Modifier.fillMaxSiz
                    color = MaterialTheme.colors.b
                ) {
                    Greeting( name: "Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    GreetingCardTheme {
        Greeting( name: "Android")
```

University of the West of England
BRISTOL

The standard main activity



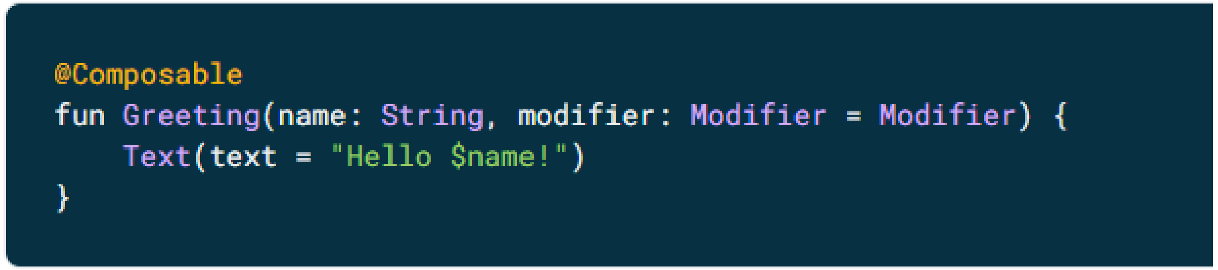
```
class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContent {
           GreetingCardTheme {
               // A surface container using the 'background' color from the theme
               Surface(
                   modifier = Modifier.fillMaxSize(),
                   color = MaterialTheme.colors.background
               ) {
                   Greeting("Android")
               }
           }
       }
   }
}
```

The standard main activity

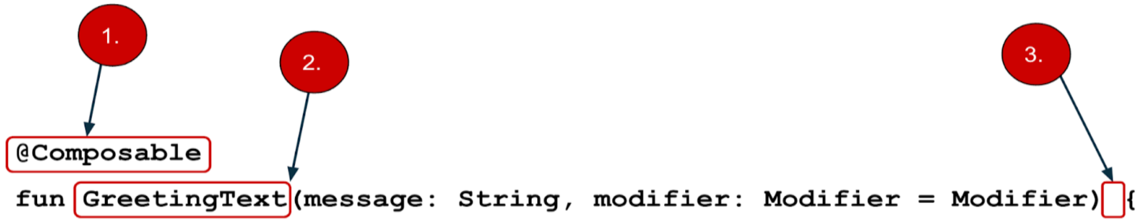
- class **MainActivity : ComponentActivity()** is the inheritance of the Compose activity.
- **OnCreate()** is the entry point, and one of the app life cycle, which is similar to main() function in Kotlin.
- The **savedInstanceState** is a reference to a Bundle object that is passed into the onCreate method of every Android Activity. The onCreate() expects to be called with a Bundle as parameter so we pass savedInstanceState.
- **Bundle?** can return null because Kotlin is null safe.
- **setContent{ }** is a function to define the layout through composable functions. It is not written like a normal function, because its last parameter is a lambda function and so { } is a block and all functions in the block are the parameters.
- The **Surface()** is a container that can be used to draw a background color or surface for other Composables. It can take modifiers, such as modifier, and specify a background color. Inside Surface, you can place other Composables that make up your app's UI.
- The Modifier class provides a set of functions that you can chain together to apply various modifications to a Composable such as

modifier = Modifier.padding(16.dp).background(Color.Blue).clickable { /* Handle click event */ }

Composable functions



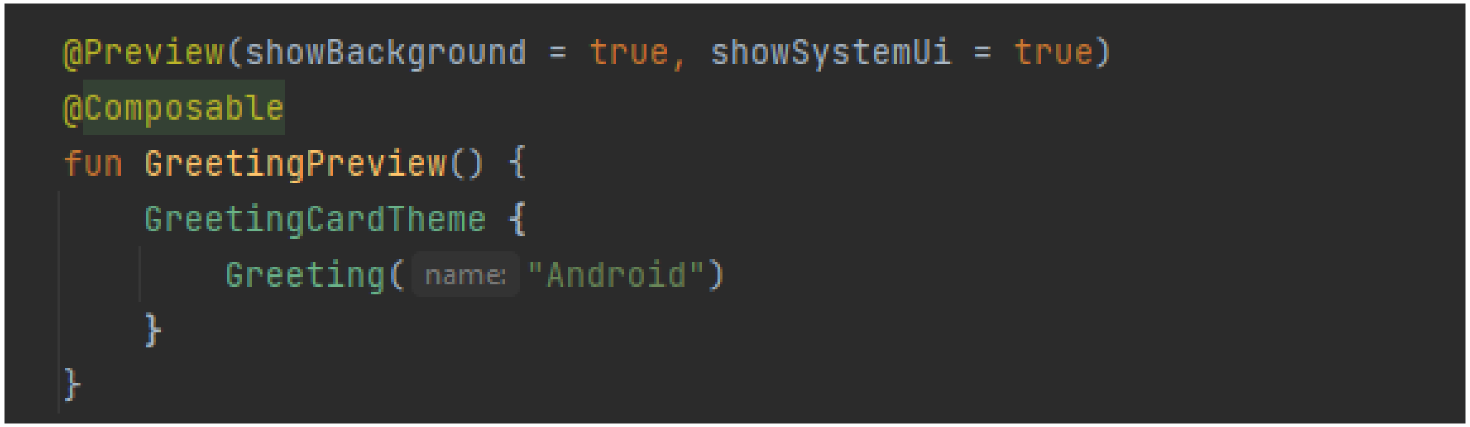
```kotlin
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(text = "Hello $name!")
}
```



```kotlin
@Composable
fun GreetingText(message: String, modifier: Modifier = Modifier) {
```

- You add the @Composable annotation before the function.
- @Composable function names are capitalized.
- @Composable functions can't return anything.
- Text is a compose component.

Preview composable functions



```
@Preview(showBackground = true, showSystemUi = true)
@Composable
fun GreetingPreview() {
    GreetingCardTheme {
        Greeting( name: "Android")
    }
}
```

- The GreetingPreview() function is a cool feature that lets you see what your composable looks like without having to build your entire app. It is independent from mainActivity().
- The @Preview annotation takes in a parameter called showBackground. If showBackground is set to true, it will add a background to your composable preview.

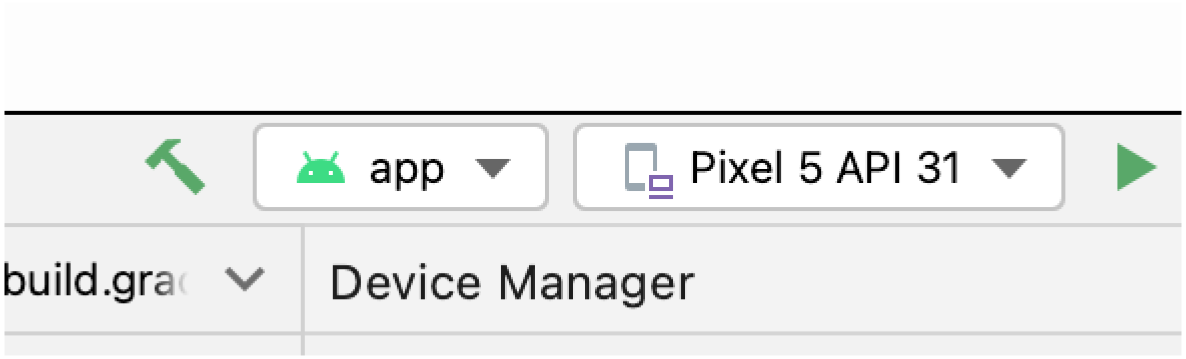
Update the text, change the background colour and add padding

```kotlin
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(text = "Hi, my name is $name!")
}
```

```kotlin
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
   Surface(color = Color.Cyan) {
       Text(
           text = "Hi, my name is $name!",
           modifier = modifier.padding(24.dp)
       )
   }
}
```

Run the app on the emulator

- You can use Device Manager to create an Android Virtual Device (AVD)
  - In Android Studio, select Tools > Device Manager, or from the right column.
- Run your app on the Android Emulator, select the virtual device from the dropdown menu and click ▶



University of the West of England
BRISTOL

How to connect your Android device

- Enable USB debugging of your device
- Install the Google USB driver (Windows only)
  - In Android Studio, click Tools > SDK Manager. The Preferences > Appearance & Behaviour > System Settings > Android SDK dialog opens.
  - Click the SDK Tools tab.
  - Select Google USB Driver and then click OK.
- When the device is connected with AS, it should be identified automatically in the device manager-> Physical.

Mighty Modifiers

- Modifiers are the fundamental building blocks for customizing and enhancing your composables in Jetpack Compose.
- It act s like decorators so that
  - change the composable's size, layout, behaviour, and appearance
  - add information, like accessibility labels
  - process user input
  - add high-level interactions, like making an element clickable, scrollable, draggable, or zoomable

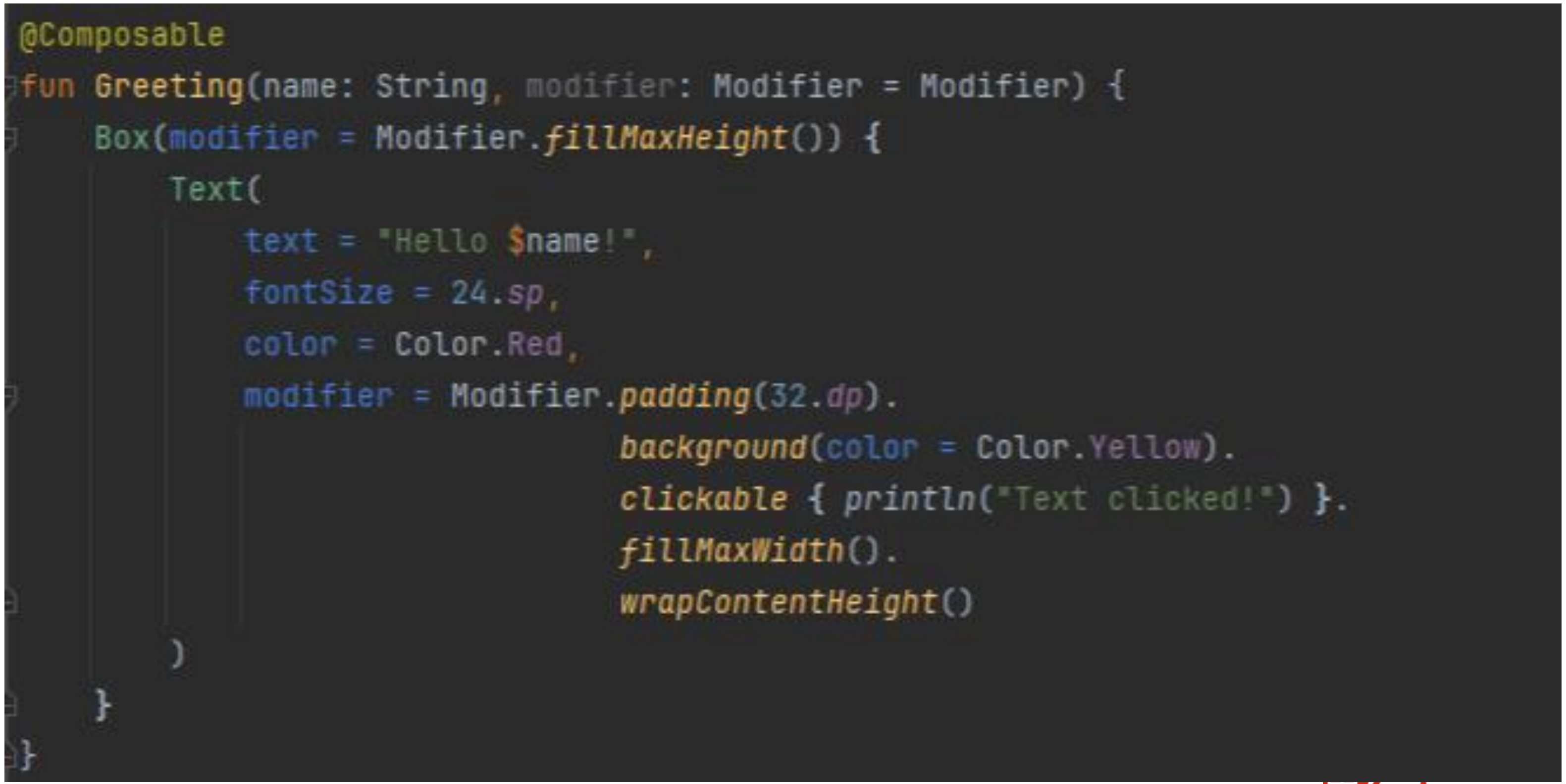
https://medium.com/@anandgaur22/modifiers-in-jetpack-compose-f2bd164fb9c6

Modifiers – controlling size, layout and appearance

- Modifier.fillMaxSize(): Acts like Match Parents in XML
- Modifier.size(400.dp): Used to give the specific size
- Modifier.fillMaxWidth(): Used for taking complete width
- Modifier.fillMaxHeight(): Used for taking complete height
- Modifier.padding(20.dp): For adding the padding
- Modifier.align(alignment = Alignment.End): For give alignment like
- Modifier.border(10.dp, Color.Yellow): used for adding border
- Modifier.background(Color.Red): for background

Modifiers – a toy example

- Change the text's size, change the text's colour, change the background's colour, Insert the padding and make the text clickable



```kotlin
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Box(modifier = Modifier.fillMaxHeight()) {
        Text(
            text = "Hello $name!",
            fontSize = 24.sp,
            color = Color.Red,
            modifier = Modifier.padding(32.dp).
                                    background(color = Color.Yellow).
                                    clickable { println("Text clicked!") }.
                                    fillMaxWidth().
                                    wrapContentHeight()
        )
    }
}
```

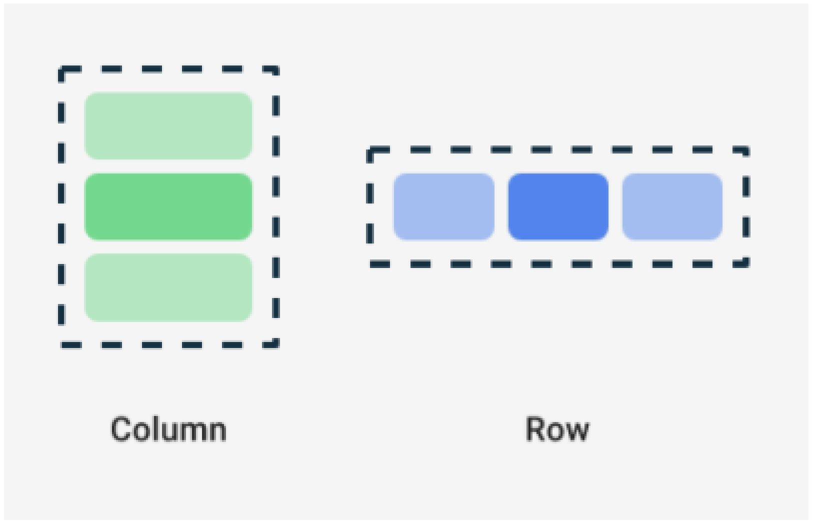
University of the West of England
BRISTOL

Basic UI layout

- The three basic, standard layout elements in Compose are Column, Row, and Box composables.

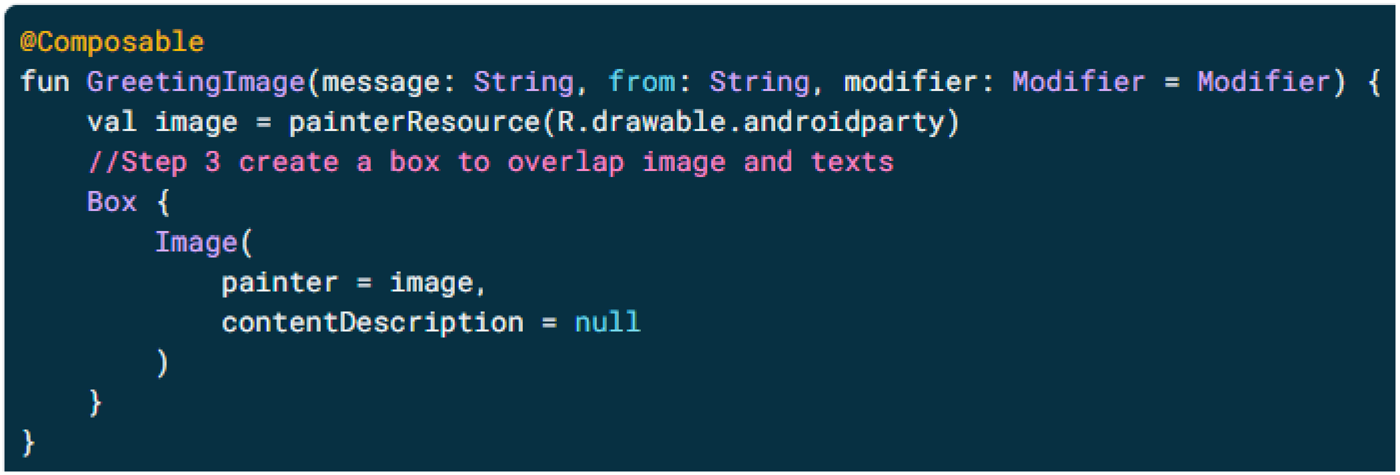
```
Row(
    content = {
        Text("Some text")
        Text("Some more text")
        Text("Last text")
    }
)
```

```
@Composable
fun GreetingText(message: String, from: String, modifier: Modifier = Modifier) {
    Row {
        Text(
            text = message,
            fontSize = 100.sp,
            lineHeight = 116.sp,
        )
        Text(
            text = from,
            fontSize = 36.sp
        )
    }
}
```



Add images into your app

- In Android Studio, click View > Tool Windows > Resource Manager or click the Resource Manager tab next to the Project window.
- Click + (Add resources to the module) > Import Drawables.
- All the imported images are stored in res->drawable
- Use painterResource() to call the images



```
@Composable
fun GreetingImage(message: String, from: String, modifier: Modifier = Modifier) {
    val image = painterResource(R.drawable.androidparty)
    //Step 3 create a box to overlap image and texts
    Box {
        Image(
            painter = image,
            contentDescription = null
        )
    }
}
```

Conclusion

- ❑ Know how to run an app via Preview, the virtual emulator and physical devices
- ❑ Understand the basic mainActivity function
- ❑ Learn how to use composable function
- ❑ Know how to use Modifers
- ❑ Know how to use basic layout UI like column and row
- ❑ Know how to add images

University of the West of England
BRISTOL