Get Started with Android Application Development Using Jetpack Compose

Week 1_2
Fall 2025

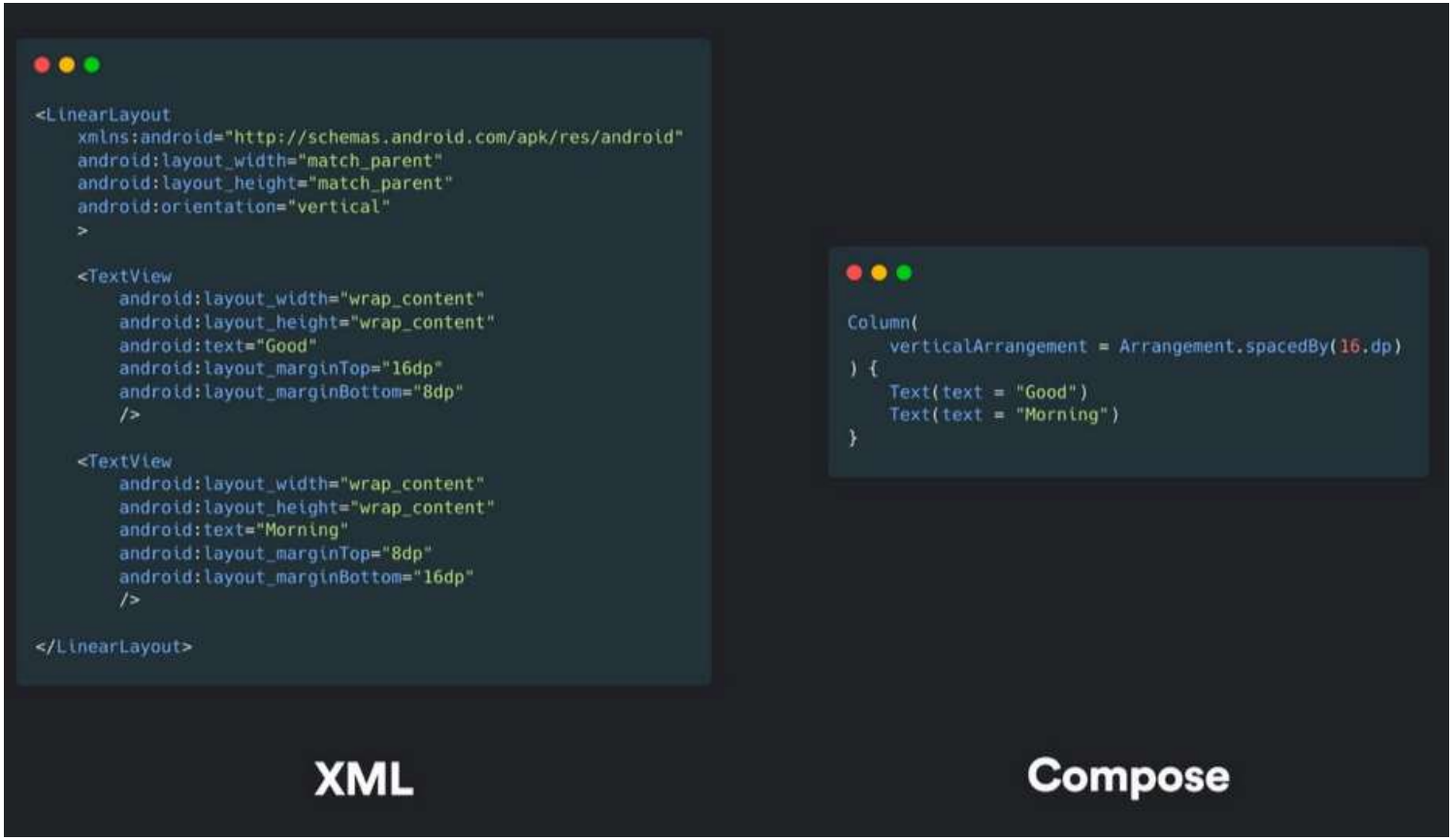
Volodymyr Voytenko

What is Jetpack Compose?

Jetpack Compose is a modern toolkit designed to simplify UI development. It combines a reactive programming model with the use of the Kotlin programming language.

* Reactive programming is a programming paradigm, or model, that centers around the concept of reacting to changes in data and events as opposed to waiting for an event to happen.

Jetpack Compose vs. XML




```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Good"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="8dp"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Morning"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="16dp"
        />

</LinearLayout>
```

```
Column(
    verticalArrangement = Arrangement.spacedBy(16.dp)
) {
    Text(text = "Good")
    Text(text = "Morning")
}
```

An xml-based UI interface

A new declarative UI toolkit for building native user interfaces

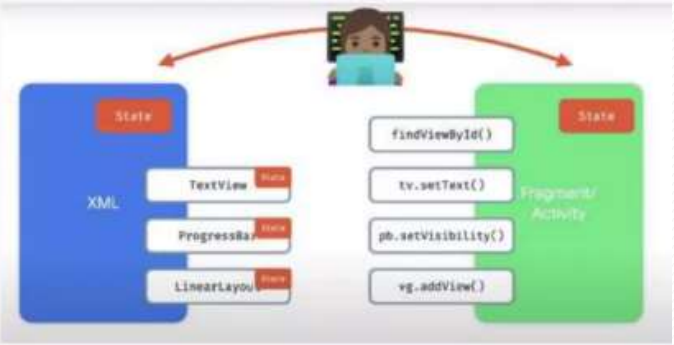
App built with Jetpack Compose

- Airbnb.
- Lyft.
- Reddit.
- Disney+ Hotstar.
- Square.
- Twitter.
- SoundCloud.
- Dropbox.

Benefit of Jetpack Compose

- Declarative
- Compatible
- Increase development speed
- Concise and Idiomatic Kotlin
- Easy to maintain
- Written in Kotlin

Imperative vs. Declarative UI



Imperative UI, the traditional approach, requires developers to specify how the UI should be created and updated explicitly.

Declarative UI focuses on describing what you want the UI to look like, and the framework takes care of how to achieve that.

Declarative UI example

Declarative UI focuses on describing what you want the UI to look like, and the framework takes care of how to achieve that.

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

// In the UI code
Greeting(name = “Alex")
```

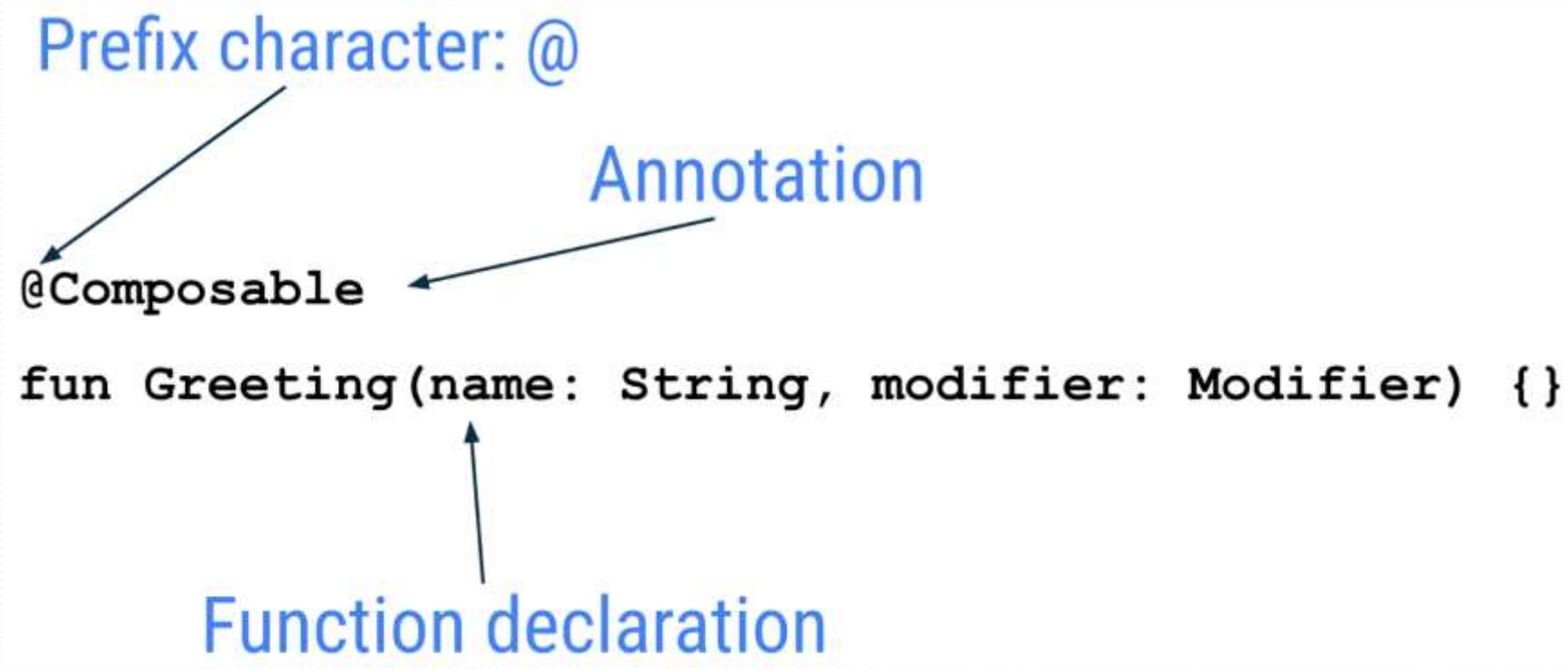
In this example, we declare a **Greeting()** as composable function – to create UI component that displays a greeting message. We specify what we want to display and let Jetpack Compose handle the rendering.
If we want to change the greeting, we simply update the name parameter, and the UI automatically updates to reflect the change.

Key Characteristics of Declarative UI:

- Reactivity: UI updates automatically when data changes.
- Component Composition: Small UI components are composed to build complex UIs.
- UI as Code: UI elements are defined in code.
- Unidirectional Data Flow: Data flows from one source to the UI.
- Simplified State Management: Built-in solutions for handling UI states.

Composable function

With Compose, we define our interfaces as functions. Composable functions are the basic building block of a UI in Compose.



Composable functions, cont


```
@Composable
fun Greeting(name: String) {
    Text("Hello $name")
}
```

•The function is annotated with the @Composable annotation.
• All Composable functions must have this annotation; this annotation informs the Compose compiler that this function is intended to convert data into UI.
•The function takes in data. Composable functions can accept parameters, which allow the app logic to describe the UI. In this case, our widget accepts a String so it can greet the user by name.
•The function displays text in the UI. It does so by calling the Text() composable function, which creates the text UI element. Composable functions emit UI hierarchy by calling other composable functions.
•The function doesn't return anything. Compose functions that emit UI do not need to return anything, because they describe the desired screen state instead of constructing UI widgets.

Note: Composable functions cannot currently be run in parallel, but you should write Compose code in a multithreaded way. In the future, Compose may be

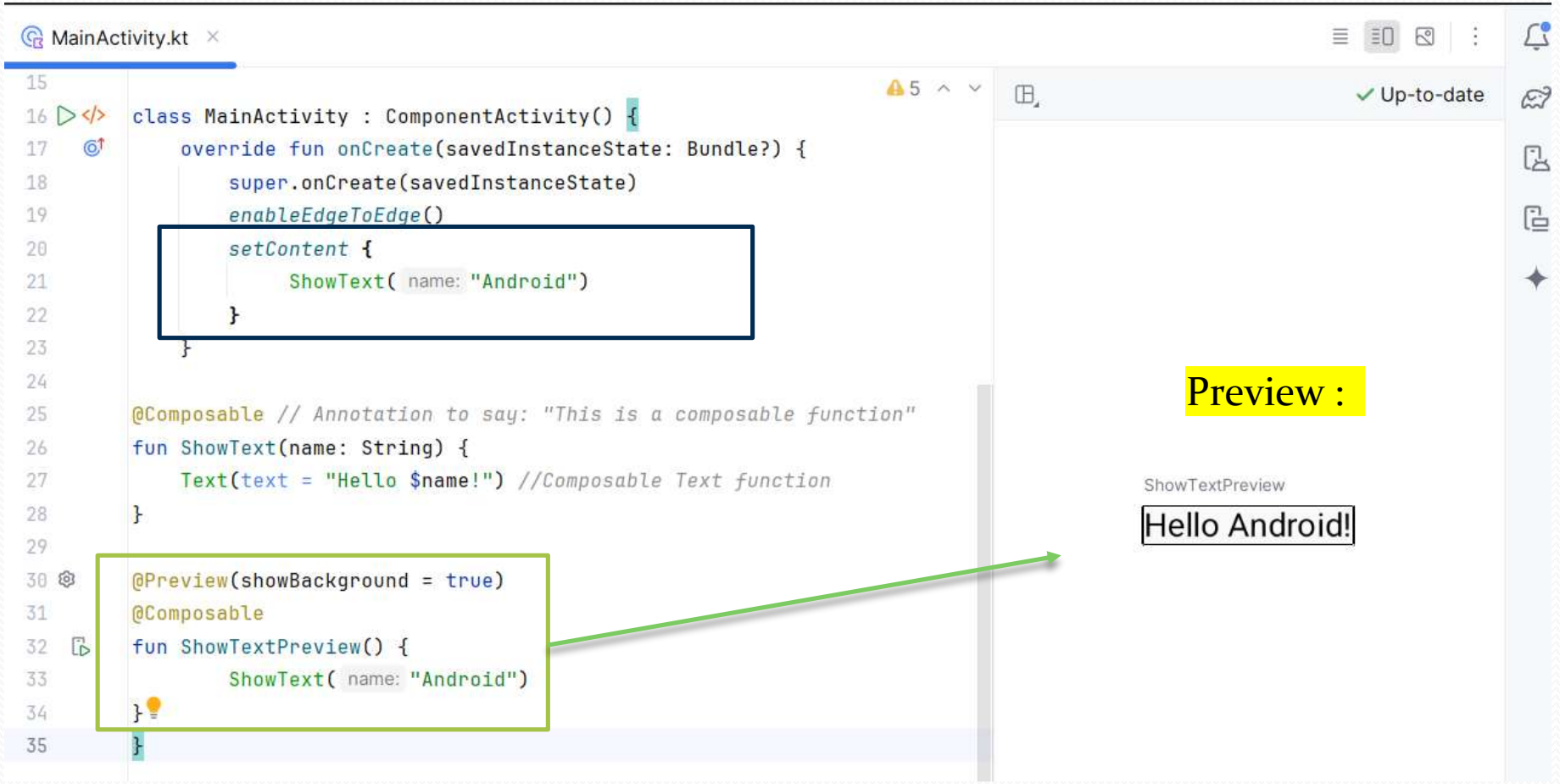
Composable preview function

To enable a preview of this composable item, we have to create another composable, annotated with **@Composable** and **@Preview**, where we call composable function :

```
44  @Preview(showBackground = true)
45  @Composable
46  fun GreetingPreview() {
47      HappyBirthdayTheme {
48          Greeting( name: "Android")
49      }
50  }
```

It will serve as a quick and convenient way to see how your UI components will look without needing to build and run the entire app on an emulator or physical device.

Add call of compose function to onCreate()



Using Modifier

Modyfing Preview

MainActivity.kt

```
10
11  class MainActivity : ComponentActivity() {
12      override fun onCreate(savedInstanceState: Bundle?) {
13          super.onCreate(savedInstanceState)
14          enableEdgeToEdge()
15          setContent {
16              ShowText( name: "Android")
17          }
18      }
19
20  @Composable // Annotation to say: "This is a composable function"
21  fun ShowText(name: String) {
22      Text(text = "Hello $name!") //Composable Text function
23  }
24
25  @Preview(showBackground = true, widthDp = 75, heightDp = 75)
26  @Composable
27  fun ShowTextPreview() {
28          ShowText( name: "Peter, Anna and all")
29  }
30  }
```

Up-to-date

Preview :

ShowTextPreview

By default, **@Preview** dimensions are chosen automatically to wrap its content. To set the dimensions manually, add **heightDp** and **widthDp** parameters. Those values are already interpreted as dp, so you don't need to add .dp to them:

Run the project!

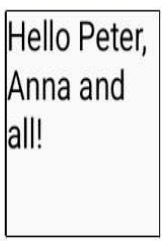
MainActivity.kt

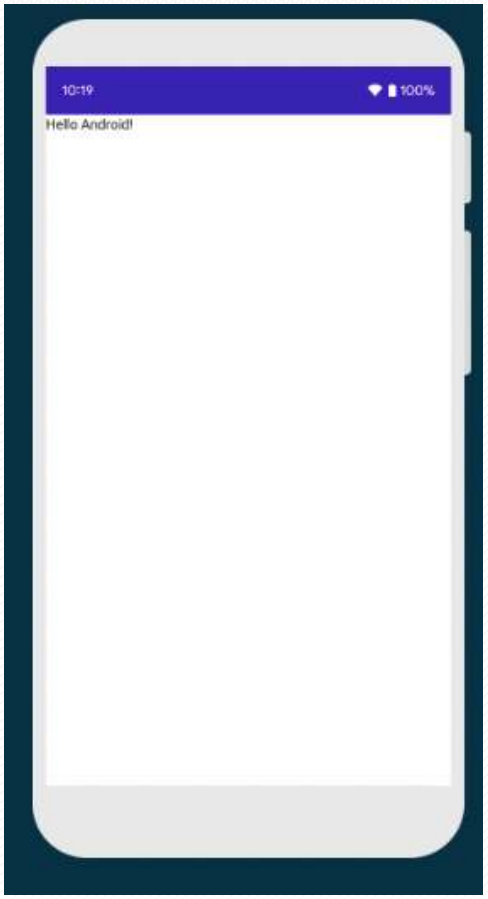
```
10
11  class MainActivity : ComponentActivity() {
12      override fun onCreate(savedInstanceState: Bundle?) {
13          super.onCreate(savedInstanceState)
14          enableEdgeToEdge()
15          setContent {
16              ShowText( name: "Android")
17          }
18      }
19
20  @Composable // Annotation to say: "This is a composable function"
21  fun ShowText(name: String) {
22      Text(text = "Hello $name!") //Composable Text function
23  }
24
25  @Preview(showBackground = true, widthDp = 75, heightDp = 75)
26  @Composable
27  fun ShowTextPreview() {
28          ShowText( name: "Peter, Anna and all")
29  }
30  }
```

Up-to-date

Preview :

ShowTextPreview





By default, @Preview dimensions are chosen automatically to wrap its content. To set the dimensions manually, add heightDp and widthDp parameters. Those values are already interpreted as dp, so you don't need to add .dp to them:

Using Modifier

In Jetpack Compose, the **Modifier** class is a powerful tool for defining and chaining UI behaviors and appearances for composable functions. It allows you to modify the layout, appearance, and interaction of composable elements without having to define custom composable.

Modifier can control the size, position, and alignment of composables:

- **fillMaxSize()**: Expands the composable to fill the maximum available size.
- **padding()**: Adds padding around a composable.
- **size()**: Sets a specific width and height for a composable.
- **offset()**: Moves the composable from its original position by a specified amount

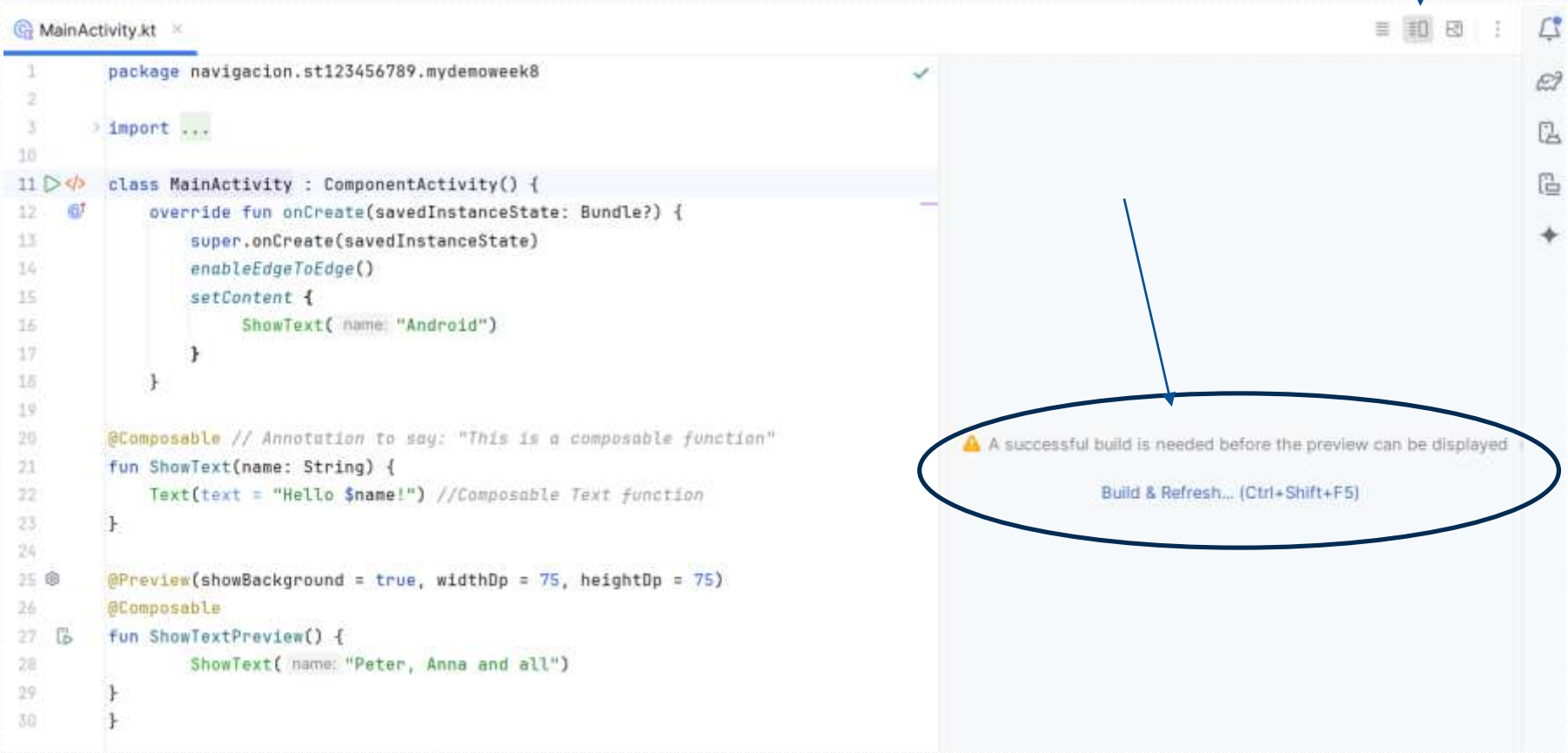
- **background()**: Adds a background color or shape.
- **border()**: Draws a border around a composable.
- **clip()**: Clips the composable into a specific shape, such as a rounded rectangle

- **align()**: Sets the alignment within a container like Box.

Your turn – update the project!

Download, unzip to your PC, open and run project to start : "DemoWeek1.zip".

Preview :



Note: a successful build is needed before preview can be displayed!

Run the app on emulator!

“ MyDemoWeek1 ” : preview mode vs run

```
import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            ShowText( name: "Android")
        }
    }

@Composable // Annotation to say: "This is a composable function"
fun ShowText(name: String) {
    Text(text = "Hello $name!") //Composable Text function
}

@Preview(showBackground = true, widthDp = 75, heightDp = 75)
@Composable
fun ShowTextPreview() {
        ShowText( name: "Peter, Anna and all")
}
}
```

Preview :

ShowTextPreview

Hello Peter, Anna and all!

Run:

The problem with text visibility will be fixed soon!

Your turn – 1 : add padding

"DemoWeekOne".

1. Modify text with padding (in dp) , using Modifier :

```
@Composable // This is a composable function
fun ShowText(name: String) {

    Text(
    text = "Hello, $name",
   // adds space around Text composables
    modifier = Modifier.padding(16.dp)
    )
```

Preview :

Note: You need to add the following imports to use Modifier , padding and unit. Press "Alt+Enter" or Option+Enter to add them for you.

```
import androidx.compose.foundation.layout.padding
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
```

Your turn – 2: modify font

2. Modify fonts property – font, size, colour, etc. :

```
@Composable // This is a composable function
fun ShowText(name: String) {

    Text(
    text = "Hello, $name",
        modifier = Modifier.padding(16.dp),
        fontFamily = FontFamily.Serif,
        color = Color.Red,
        fontSize = 20.sp
        )
  }
@Preview(showBackground = true)
@Composable
fun ShowTextPreview() {
        ShowText( name: "Peter and Anna!")
}
```

Preview :

You will be prompted to add new "import" statements (" Alt + Enter") :

```
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.unit.sp
```

Your turn – 3 : modify background

3. Apply a colored background. For a rounded background, you can use Modifier.background with shape to create rounded corners.

```
@Composable // This is a composable function
fun ShowText(name: String) {

    Text(
    text = "Hello, $name",
        modifier = Modifier
            .background(Color.Yellow, shape = RoundedCornerShape(8.dp))
            .padding(16.dp),
        fontFamily = FontFamily.Serif,
        color = Color.Red,
        fontSize = 20.sp
        )
}
```

Preview :

ShowTextPreview

Hello, Peter and Anna!

You will be prompted to add new “import” statements (“ Alt + Enter”) :

```
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
```

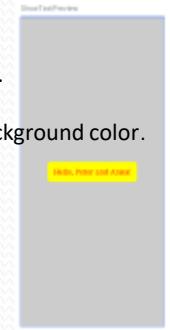
Your turn – 4: add to Box with alignment

4. Add our Text to Box with size, background and alignment, to display at the center of the screen.

```
@Composable // This is a composable function
fun ShowText(name: String) {

    Box(
        modifier = Modifier
            .fillMaxSize() // Make the Box fill the entire screen
            .background(Color.LightGray), // Optional background color for the wh
        contentAlignment = Alignment.Center // Center content within the Box
    )
    {

        Text(
            text = "Hello, $name",
            modifier = Modifier
                .background(Color.Yellow, shape = RoundedCornerShape(8.dp))
                .padding(16.dp),

            fontFamily = FontFamily.Serif,
            color = Color.Red,
            fontSize = 20.sp
            )
    }
}
```

- **Box**: It acts as a container, filling the screen and aligning its content in the center.
- **contentAlignment**: Setting this to Alignment.Center centers the Text inside the Box.
- **Modifier.fillMaxSize()**: Ensures the Box fills the entire screen size.

This way, the text appears centered on the screen with the specified padding and background color.



You will be prompted for new "import" statement ("ALT + ENTER") :

```
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.ui.Alignment
```

References

- https://developer.android.com/kotlin/compose
- https://medium.com/androiddevelopers/composed
- ANDROID PROGRAMMING: THE BIG NERD RANCH GUIDE, 4TH EDITION , by Bill Phillips (Author), Chris Stewart (Author), Kristin Marsicano (Author)
- http://developer.android.com/guide/components/
- http://www.lynda.com/Android-tutorials/Building-Adaptive-Android-