BCS 371 Mobile Application Development I

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Basic Compose GUI
  - Composable functions
  - setContent
  - Text
  - TextField
  - it keyword
  - Button
  - Column
  - Row
  - Modifier
  - Surface

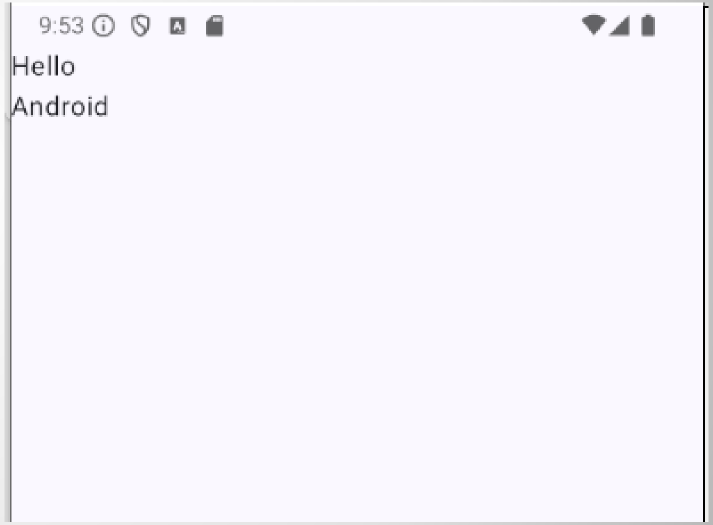
Composable Function

- Composable functions emit UI elements.
- Used to define the app's UI.
- A Composable function must be decorated with the @Composable annotation.
- For example:

Composable annotation

```
@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        Text(text = "Hello")
        Text(text = "Android")
    }
}
```

Both Column and Text are composable functions. They make up the UI elements of MainScreen.



Hello
Android

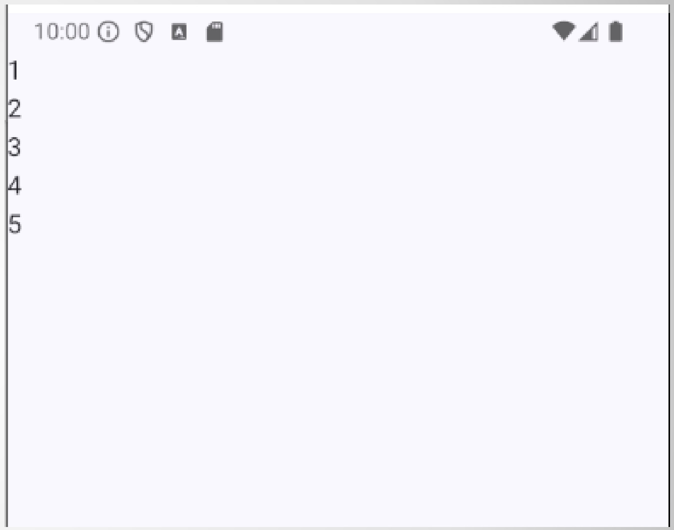
Composable Function

Create Composables in a Loop

- You can use a loop to generate multiple UI elements.

```
@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        for (i in 1..5) {
            Text(text = i.toString())
        }
    }
}
```

Creates Text elements in a loop. The Column will create five Text elements inside of it. Each Text will display a different value of i from 1 to 5.



```
1
2
3
4
5
```

Create Composables in a Loop

Calling Composable Functions

- Composable functions CANNOT be called from normal Kotlin functions.
- In the code below, MainScreen cannot be called from inside of MainActivity.onCreate (normal Kotlin function).

onCreate is a normal Kotlin function

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        ~~MainScreen(Modifier)~~
    }
} // end - MainActivity
```

Cannot call a composable function from a normal Kotlin function (onCreate is normal Kotlin function)

```
@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        Text(text = "Hello")
        Text(text = "Android")
    }
}
```

MainScreen is a composable function because it is decorated with @Composable. It CANNOT be called from a normal Kotlin function.

Calling Composable Functions

setContent Block

- Defines an activity's layout where composable functions are called.
- Bridges the gap between normal functions and composable functions.
- Composable functions can be called by setContent (composable functions emit UI).
  - setContent itself is not a composable function.
- Here is setContent in MainActivity:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            // Add GUI elements here...
        }
    }
}
```

Composable functions can be called from inside of setContent

setContent Block

Call Composable Function from setConent (Scaffold and enableEdgeToEdge)

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            HelloWorldTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    MainScreen(Modifier.padding(innerPadding))
                }
            }
        }
    }
}
@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        Text(text = "Hello")
        Text(text = "Android")
    }
}
```

MainScreen is called from inside of MainActivity's setContent block.
Note: enableEdgetoEdge is being used with a Scaffold so padding must be set on MainScreen. This will ensure that the existing content at the top of the screen (time, battery icon etc…) does not obscure parts of the UI generated by MainScreen.

Column uses modifier parameter (this will add padding)

Call Composable Function from setContent (Scaffold and enableEdgeToEdge)

Call Composable Function from setConent (No Scaffold or enableEdgeToEdge)

- For example:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
        super.onCreate(savedInstanceState)
        setContent {
            MainScreen(Modifier)
        }
    }
}

@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        Text(text = "Hello")
        Text(text = "Android")
    }
}
```

MainScreen is called inside of MainActivity's setContent block (it will be the Ui for MainActivity)

Call Composable Function from setContent (No Scaffold or enableEdgeToEdge)

Text

- A composable function.
- Shows text in the GUI.
- Must be called from a composable function or from inside a setContent block.

- For example:

```
Text(text="Hello")
```

Emits the string "Hello" in the UI

Text

Text That Uses a Variable Source

- Put data from a variable in a Text.
- For example:

Declare variable x. Using rememberSaveable and mutableStateOf will keep the variable value so it can be used whenever the UI is recomposed (initial value of x is "abc")

```
var x by rememberSaveable { mutableStateOf("abc") }

Text(text="$x")
```

Use $ prefix for a variable name.

$x will be replaced with whatever value is in the x variable.

Note: rememberSaveable saves data longer than remember. Check next slide for more information.

Text That Uses a Variable Source

<u>remember vs rememberSaveable</u>

- **remember** – Data is retained through recompositions. Data is NOT retained through a configuration change (the containing activity is destroyed during a configuration change).

- **rememberSaveable** – Data is retained through recompositions and data is retained through a configuration change.

- rememberSaveable retains data in more circumstances than remember.

remember vs rememberSaveable

TextField

- A composable function
- Allows user to input text.
- Must be called from a composable function or from inside a setContent block.

- Make sure to use at least version 1.1.2 of the material3 dependency in the Gradle (app) file (make sure to Sync the Gradle file):

```
implementation("androidx.compose.material3:material3:1.1.2")
```

$\uparrow$

Add the dependency version number

TextField

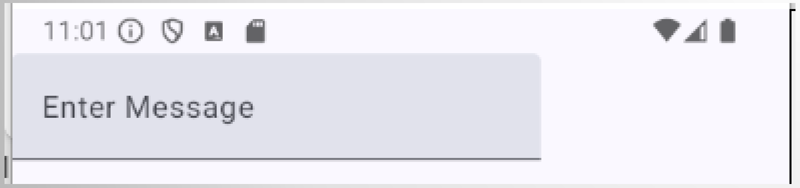
TextField

- When creating a TextField you must also declare a variable to hold the data type in the TextField.

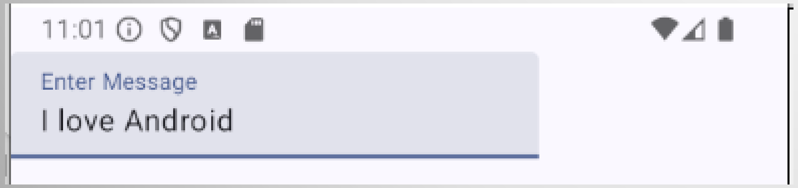
The text variable stores the string value being displayed in the TextField

```
var text by rememberSaveable { mutableStateOf("") }
TextField(
    value = text,
    onValueChange = { text = it },
    label = { Text("Enter Message") }
)
```

onValueChange. This event handler runs if the value in the TextField changes. This code will update the text variable value so that it matches the value in the TextField.
Note: The it keyword is described on an upcoming slide.



"Enter Message" is the label for the TextField



I love Android

When the user starts typing in the TextField the label will shrink. Any data typed in the TextField goes in the text variable.

TextField

Create TextField in Function

- Define a function that creates both the TextField and its associated variable.

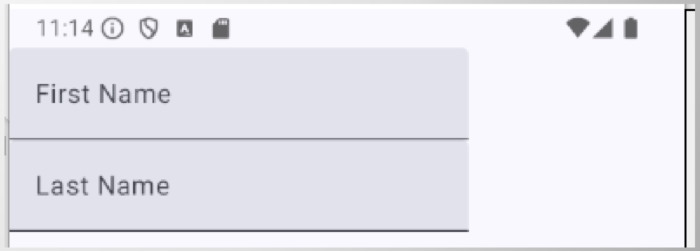
```
@Composable
fun SimpleFilledTextFieldSample(labelToUse: String) {
    var text by rememberSaveable { mutableStateOf("") }

    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text(labelToUse) }
    )
}
```

Each TextField created by this function will have its own text variable instance (typing in one TextField will not affect any of the others generated by this function).

label is passed as a parameter

```
@Composable
fun MainScreen(modifier: Modifier) {
    Column(modifier) {
        SimpleFilledTextFieldSample("First Name")
        SimpleFilledTextFieldSample("Last Name")
    }
}
```

Call the SimpleTextField function and pass in the label to use for it



Create TextField in Function

it Keyword

- Use with a one argument lambda.
- You can omit the parameter definition if there is only one parameter and just use "it".
- In the code below onValueChange takes one parameter which will contain the new value typed in the TextField).
- This parameter is omitted in favor of using "it" instead. So "it" will contain the new value typed in the TextField.

```
@Composable
fun SimpleFilledTextFieldSample() {
    var text by rememberSaveable { mutableStateOf("") }

    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("Label") }
    )
}
```

Use "it" keyword instead of formally defining the one and only parameter to onValueChange.

The following code is equivalent:
onValueChange = { value → text = value}
This version of the lambda names the one and only parameter "value".

it Keyword

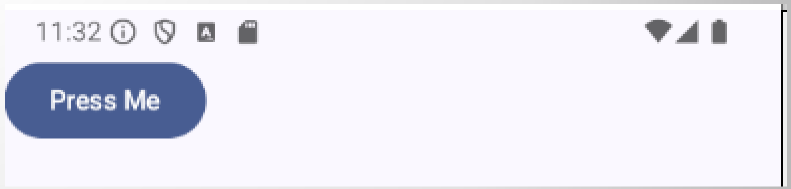
Button

- A composable function.
- Allows the user to trigger an action.
- For example:

onClick is the event handler for the Button. When the Button is pressed it will run the code in the function associated with onClick.

```
Button(
    onClick = {
      // Do something like set a value or show a toast here…
    }
)
{
    Text(text="Press Me")
}
```

Text that is displayed on the button itself



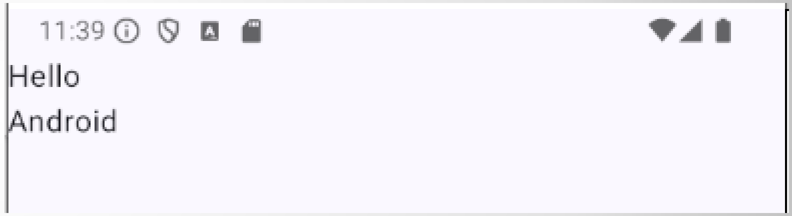
Button

Column

- A composable function.
- Arrange multiple UI items one after another vertically.

- For example:

```
Column(modifier) {
    Text(text = "Hello")
    Text(text = "Android")
}
```



Hello
Android

Column arranges items vertically

Column

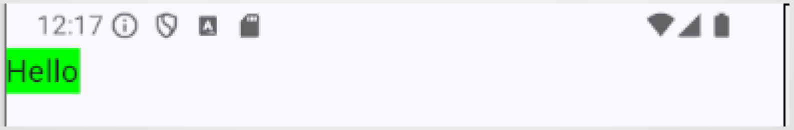
Modifier

- Modifiers allow you to decorate or augment a composable.
- Create a Modifier instance and pass it to a composable.
- Here are some things that modifier can be used on:
  - Set color
  - Set size
  - Set padding
  - Set event handlers
- Here is an example usage:

Text("Hello", modifier = **Modifier.background(Color.Green)**)

Create a new instance of the Modifier class

Call background function on the Modifier instance and pass in the color green



- Link: https://developer.android.com/develop/ui/compose/modifiers

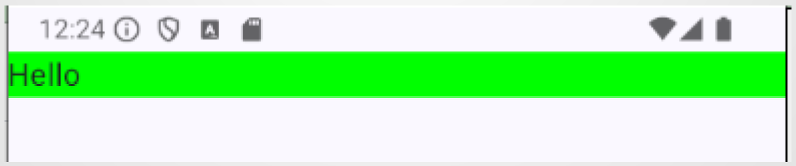
Modifier

Chain Function Calls on Modifier Instance

- Modifier member functions return an instance of Modifer.
- A consequence of this is we can now chain together multiple function calls (as many as we want).
- For example:

```
Text("Hello",
    modifier = Modifier
        .background(Color.Green)
        .fillMaxWidth()
)
```

This will set the background to green and make it fill the width of the screen (to the limit of the parent container)



Chain Function Calls on Modifier Instance

Use Existing Modifier Instance

Use Existing Modifier Instance

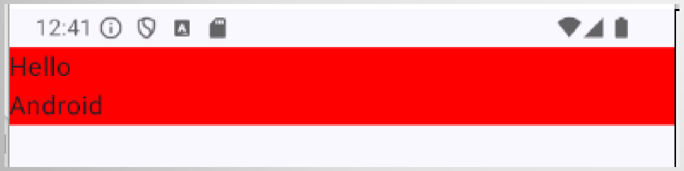
- You use an existing Modifier instance and configure it more.
- If it is passed as a parameter, then it may have some settings that need to be retained.
- For example:

```
@Composable
fun MainScreen(modifier: Modifier) {
    Column(
        modifier
            .background(Color.Red)
            .fillMaxWidth()
    ) {
        Text("Hello")
        Text("Android")
    }
}
```

This uses modifier (lowercase m). It is using the parameter (not a new instance). The background and fillMaxWdith are being added to it.

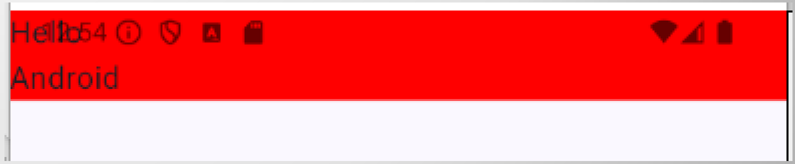
A scenario where this is useful is when MainScreen is called from inside a Scaffold with enableEdgeToEdge turned on. The Modifier instance passed into MainScreen should have the padding set on it so that the top of screen content does not overlap with it. Column will then use that padding as well as the background and fillMaxWidth settings (see pics below).

Using modifier (lowercase)


12:41

Hello
Android

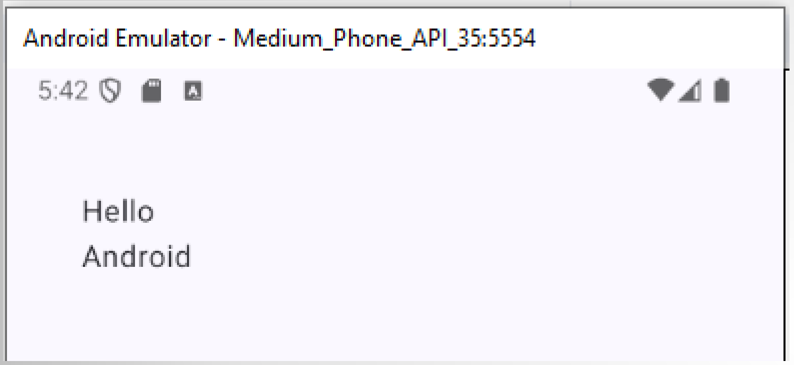
Using Modifier (uppercase) so no padding


12:54

Hello
Android

Surface

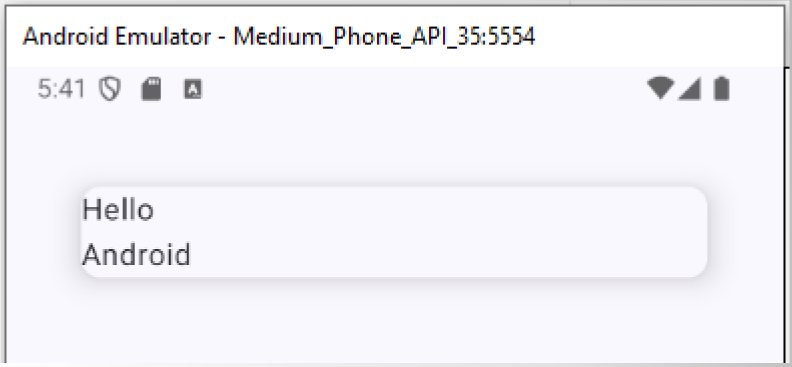
- Material surface is the central metaphor in material design.
- Each surface exists at a given elevation, which influences how that piece of surface visually relates to other surfaces and how that surface casts shadows.


Android Emulator - Medium_Phone_API_35:5554

5:42

Hello
Android

```
Surface (
    modifier = Modifier.fillMaxWidth().padding(40.dp),
) {
    Column {
        Text("Hello")
        Text("Android")
    }
}
```


Android Emulator - Medium_Phone_API_35:5554

5:41

Hello
Android

```
Surface (
    modifier = Modifier.fillMaxWidth().padding(40.dp),
    shape = RoundedCornerShape(10.dp),
    shadowElevation = 10.dp,
    onClick = {
        Toast.makeText(context, "Surface was clicked",
          Toast.LENGTH_SHORT).show()
    }
) {
    Column {
        Text("Hello")
        Text("Android")
    }
}
```

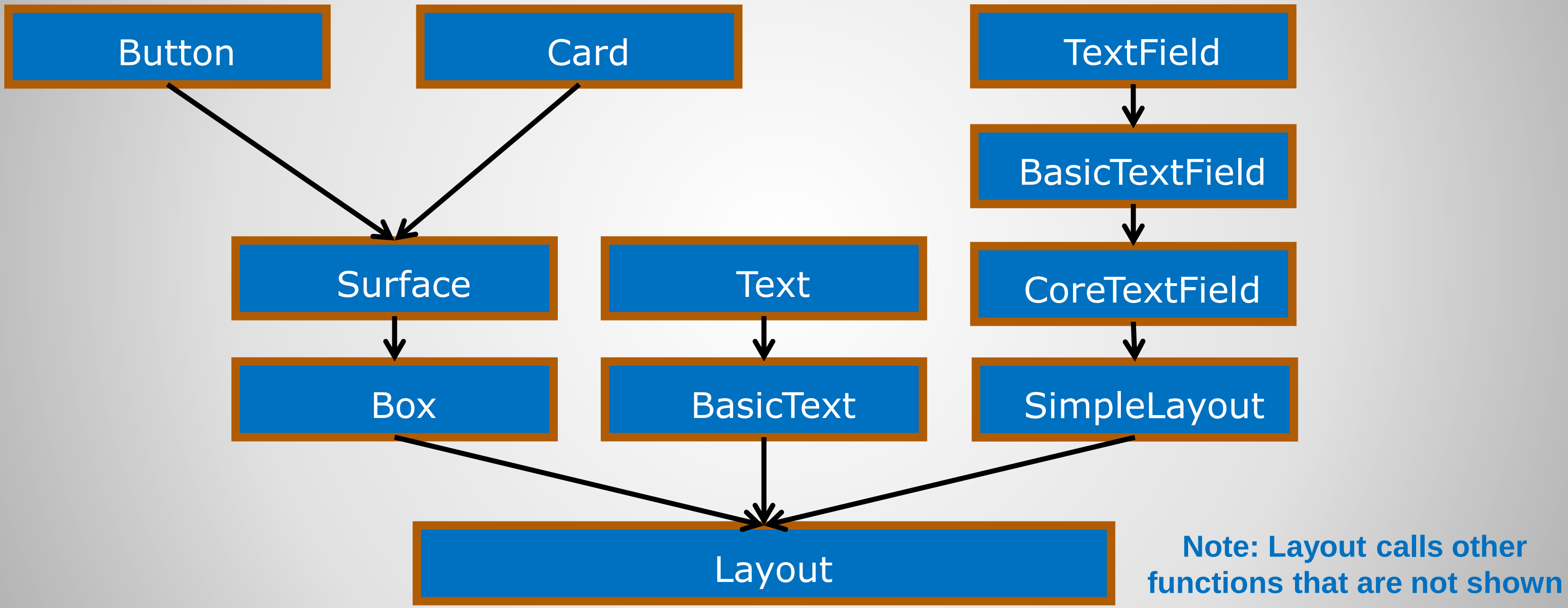
This surface uses shape, shadowElevation and is clickable

Definition taken from: https://developer.android.com/reference/kotlin/androidx/compose/material/package-summary

Surface

Composable Function Relationships

- The prebuilt composable functions call other composable functions in their implementation.
- The arrow means one composable calls another. For example, the Button function calls the Surface function.



Composable Function Relationships

End of Slides

- End of Slides