BCS 371 Mobile Application Development I

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

Today’s Lecture

- Jetpack Compose
  - Composition
  - Lifecycle of Composables
  - Activity lifecycle

Jetpack Compose

Jetpack Compose

- Declarative UI framework.
  - Declarative - Describe what you want the final result to be as opposed to describing exactly how to do it.
- The tech industry as a whole is moving towards declarative UI (for example React, SwiftUI).

Composable Function

Composable Functions

- Composables are the fundamental building blocks of apps built with Jetpack Compose.
- Use @Composable annotation above a function to create a composable function.
  - This annotation informs the Compose compiler that the function will be converting data to UI.
- Composable functions should not return values.

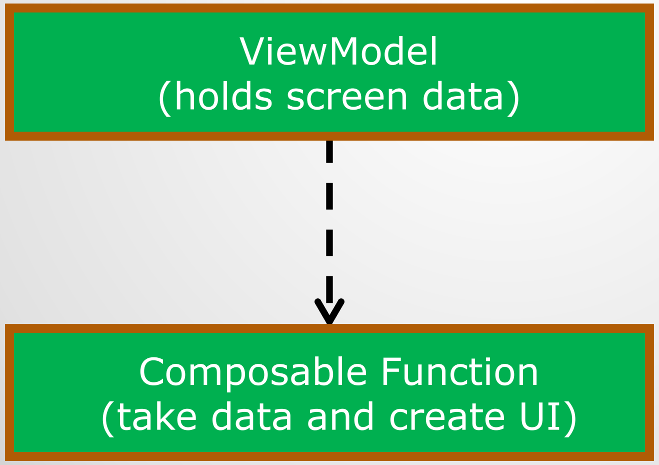
Pass data to display as a parameter

```
@Composable
fun ShowMessage(message: String) {
    Text( text = "The message is $message" )
}
```

Uses the message variable in Text.
Use a variable in a string by
prefixing the variable name with $.

Composable Functions

- In a properly designed app, the data a composable function displays should come from a screen's ViewModel.
  - A ViewModel manages data for one screen (each screen has its own ViewModel).
  - More on ViewModel later in the course.
- Composable functions should not produce "side effects".
  - For example, it should not write to a property of a shared object.
  - Instead, it should call functions on the ViewModel and have the ViewModel update data.

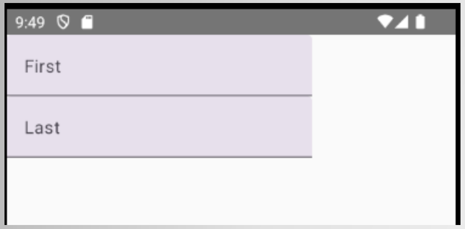


Composable function data should mainly come from the ViewModel

Composable Function

Composition

- A composition describes the UI of an app.
- It is a tree structure that consists of composables.

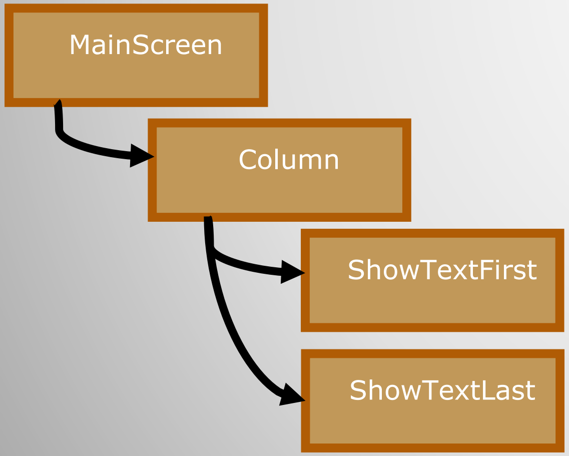


```
@Composable
fun MainScreen() {
    println("MainScreen called")
    Column {
        println("Column called")
        ShowTextFieldFirst()
        ShowTextFieldLast()
    }
}
```

```
@Composable
fun ShowTextFieldFirst() {
    println("ShowTextFieldFirst called")
    var text by rememberSaveable { mutableStateOf("") }
    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("First") }
    )
}
```

```
@Composable
fun ShowTextFieldLast() {
    println("ShowTextFieldLast called")
    var text by rememberSaveable { mutableStateOf("") }
    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("Last") }
    )
}
```

Composition (tree of composables)



Composition

Composition

- UI controls are static and must be regenerated to change their values.
- If a value in a UI control changes after the initial composition, it must perform a recomposition.
- The old view-based UI controls allowed state changes, but this has been removed in Compose.
  - Old view-based UI controls had get/set methods that allowed the state to be changed.
  - Manipulating views manually with get/set was deemed to be more error prone.

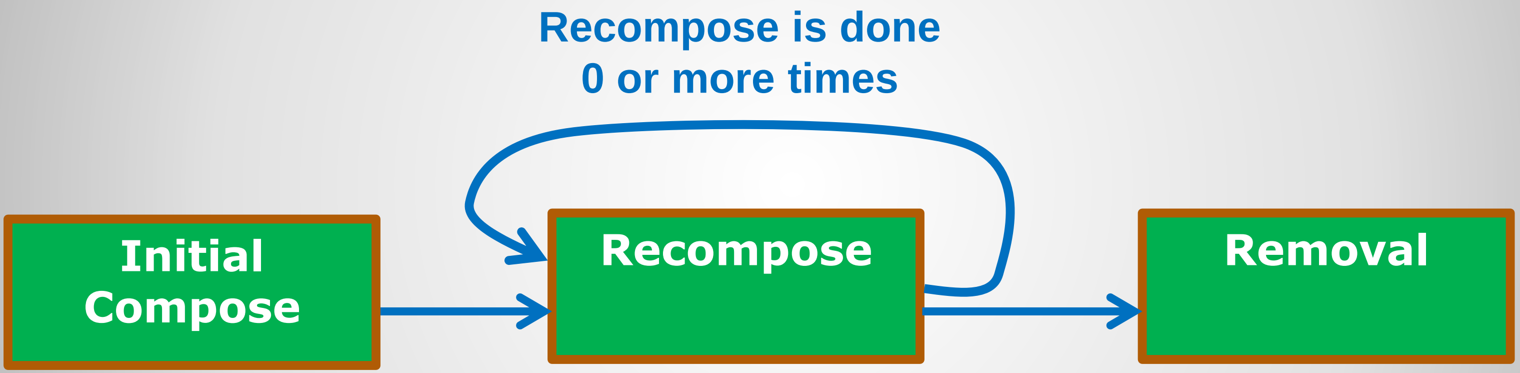
- Content adapted from the following link:

https://developer.android.com/jetpack/compose/mental-model

Composition

Lifecycle of Composables

- Initial composition happens once.
- Recomposition is generally triggered by a change in state and can happen 0 or more times.



- Taken from: https://developer.android.com/jetpack/compose/lifecycle

Lifecycle of Composables

Composition and Battery Usage

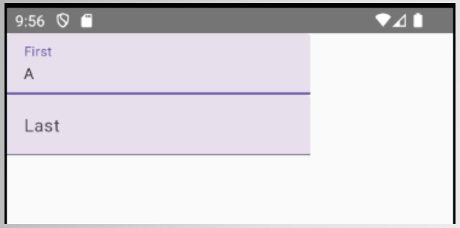
- Important! Compose only regenerates parts of the UI where an update has occurred since the previous composition.
- This is important because it would be computationally expensive to regenerate the whole UI during each recomposition.
- Less computational work means it will require less power so battery usage will be minimized.

- For example…

Composition and Battery Usage

Composition and Updates

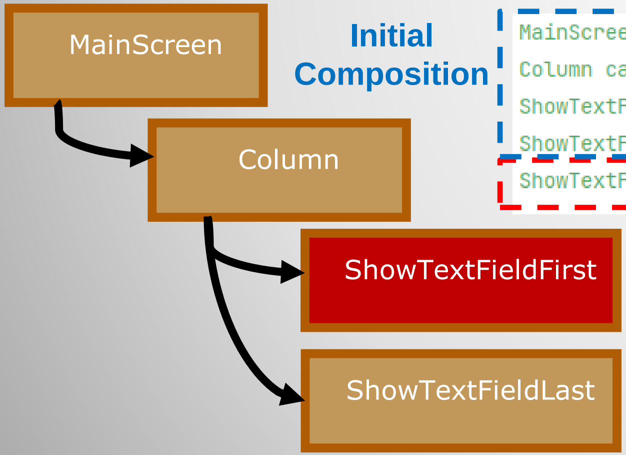
- The user types "A" in First which triggers a recomposition.
- Only ShowTextFieldFirst is called (the other functions are not called).



```
@Composable
fun MainScreen() {
    println("MainScreen called")
    Column {
        println("Column called")
        ShowTextFieldFirst()
        ShowTextFieldLast()
    }
}
```

```
@Composable
fun ShowTextFieldFirst() {
    println("ShowTextFieldFirst called")
    var text by rememberSaveable { mutableStateOf("") }
    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("First") }
    )
}
```

Composition (tree of composables)



Initial Composition

```
MainScreen called
Column called
ShowTextFieldFirst called
ShowTextFieldLast called
```

```
ShowTextFieldFirst called
```

After A is typed a recomposition occurs. It only calls ShowTextFieldFirst during recomposition.

```
@Composable
fun ShowTextFieldLast() {
    println("ShowTextFieldLast called")
    var text by rememberSaveable { mutableStateOf("") }
    TextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("Last") }
    )
}
```

Composition

Lifecycle - LaunchedEffect

- LaunchedEffect - Composable that is executed when the initial composition happens.

```
LaunchedEffect(Unit) {

    // Add code to run during the initial composition

}
```

Note: Unit in Kotlin is similar void in Java. Use it when a function does not return a meaningful value.

LifeCycle - LaunchedEffect

Lifecycle - SideEffect

- SideEffect - Composable that is executed when a composition happens (either an initial compose or a recompose).

```
SideEffect {

    // Add code to run during a composition

}
```

LifeCycle - SideEffect

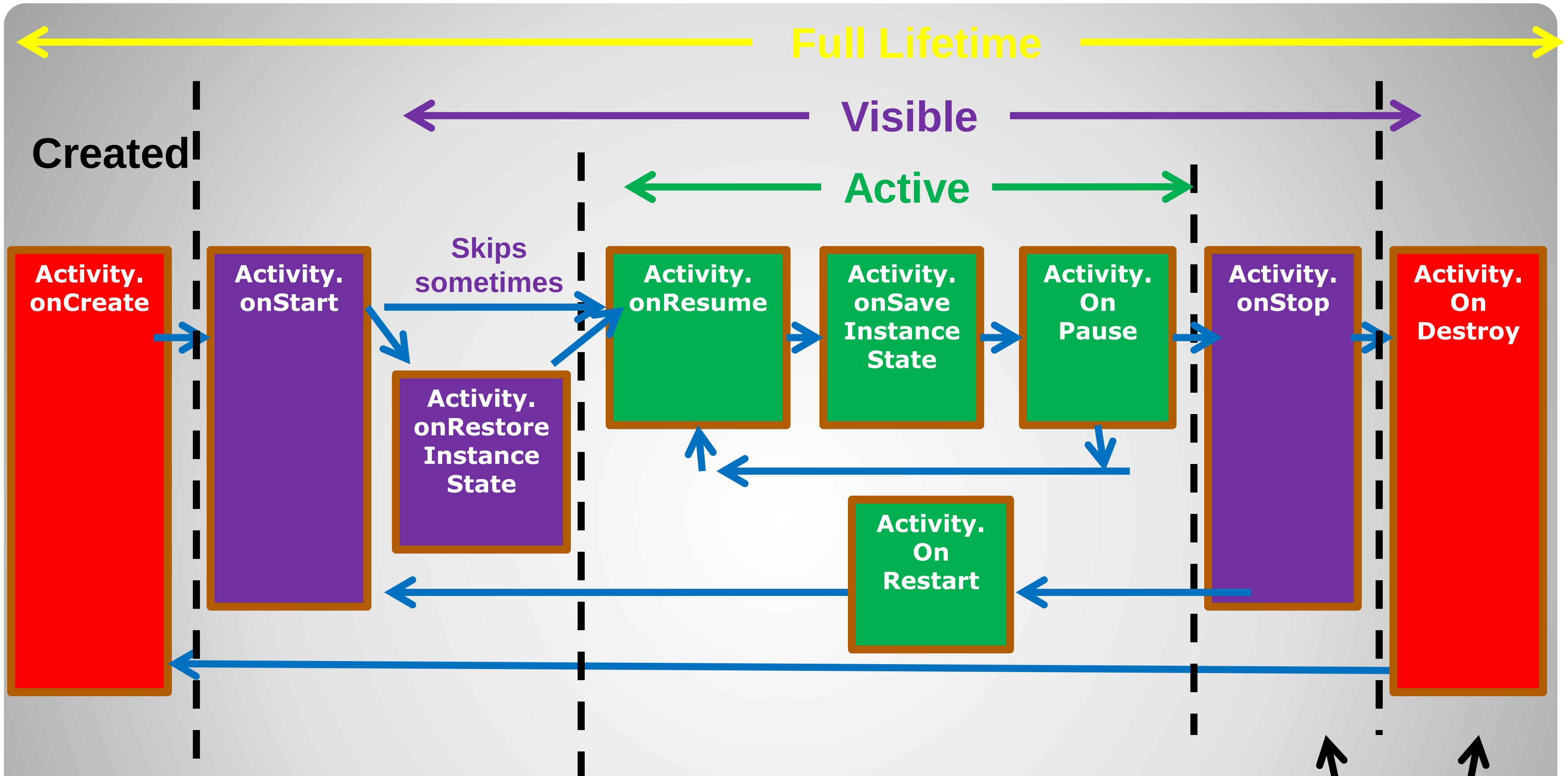
- Now on to the activity lifecycle…

Activity LifeCycle

Activity Lifecycle

- The activity that contains all the composable functions has its own lifecycle events.
- Some activity lifecycle events:
  - onStart (becomes at least partially visible)
  - onResume (in foreground and user can interact with)
  - onStop (not visible)
  - onPause(partially visible, not in foreground)
- The app can react to these events and optimize its use of the device's resources.
- For example, the app can stop some computations when it is not in the foreground and then restart those computations when it moves back to the foreground.
- A composable function can be setup to run code when these activity lifecycle events happen.
- Note: If the containing activity is destroyed the lifecycle event handlers will not execute.

Activity LifeCycle



Activity Lifecycle

Activity LifeCycle Event Handlers

Activity Lifecycle Event Handlers

- Use lifecycle effect functions to run code when the activity lifecycle events occur.
- For example:

```
@Composable
fun MainScreen() {
    LifecycleEventEffect(Lifecycle.Event.ON_PAUSE) {
        println("Lifecycle ON_PAUSE")
    }
    LifecycleEventEffect(Lifecycle.Event.ON_RESUME) {
        println("Lifecycle ON_RESUME")
    }
    LifecycleEventEffect(Lifecycle.Event.ON_START) {
        println("Lifecycle ON_START")
    }
    LifecycleEventEffect(Lifecycle.Event.ON_STOP) {
        println("Lifecycle ON_STOP")
    }
}
```

← Runs when ON_PAUSE event occurs

← Runs when ON_RESUME event occurs

← Runs when ON_START event occurs

← Runs when ON_STOP event occurs

- End of Slides

End of Slides