BCS 371 Lab – State and State Hoisting

Overview

Write an app that saves state in a local variable of a composable function. Implement state hoisting.

Create a project

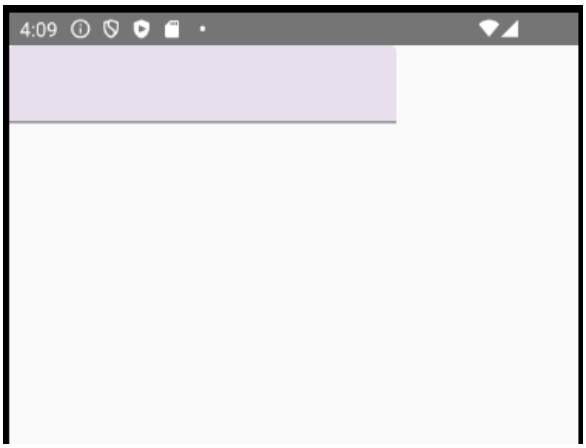
Create a new Android application in Android Studio. Choose the **Empty Activity** type to create an empty activity that uses Jetpack Compose.

Setup the Main Screen

Create a Kotlin file named MainScreen.kt. Add a composable function named MainScreen. Here is the function header:

```
@Composable
fun MainScreen(modifier: Modifier = Modifier)
```

It should look like the following (only one TextField):



More specifications for this composable function:

- Declare the following variable to be used by the TextField at the top of the function:
  `var name = ""`
- Set the TextField's value and onValueChange to the following:
  ```
  TextField(
     value = name,
     onValueChange = { name = it },
     modifier
  )
  ```
- Add a SideEffect block to MainScreen. Print the following message to the Logcat window in the SideEffect block: "SideEffect - Composition executed"

- Update MainActivity to call MainScreen.

Run the App

Type in the TextField. The data in the TextField will not change because the variable name does not send out a notification that it has changed. No recomposition happens because it does not think the name variable value has changed (the screen remains the same). Check the Logcat window and make sure that there is only one "SideEffect - Composition executed" message (for the initial composition).

Update MainScreen – Use mutableStateOf

Update MainScreen so that the name variable uses mutableStateOf in its declaration. Change it to the following:

```
var name by mutableStateOf("")
```

Note: You will see an error message in Android Studio stating that you are using a state object during composition without using remember. Ignore this error for now.

Run the App

Type in the TextField. The data in the TextField still has not changed. A notification is being sent out when the name variable changes so recomposition occurs each time a letter is typed at the keyboard. The problem now is that the value of the name variable is lost during recomposition (it does not remember the value through the recomposition). Check the Logcat window and make sure that a "SideEffect - Composition executed" message appears each time a letter is typed into the TextField.

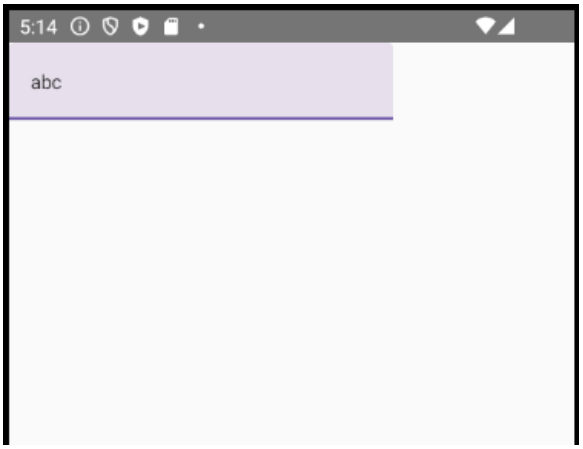
Update MainScreen – Use remember and mutableStateOf

Update MainScreen so that the name variable uses both remember and mutableStateOf in its declaration. Change it to the following:

```
var name by remember { mutableStateOf("") }
```

Run the App

Type in the TextField. The data in the TextField should now change as letters are typed on the keyboard. Check the Logcat window and make sure that "SideEffect - Composition executed" appears each time a letter is typed into the TextField. Here is a screenshot after typing "abc":



Now try rotating the device (you might need to turn on Auto-rotate on the device). It will show the following:

When the device is rotated it causes a configuration change on the device. This will cause the activity to be destroyed and recreated. Using remember is not enough to retain data through a configuration change.

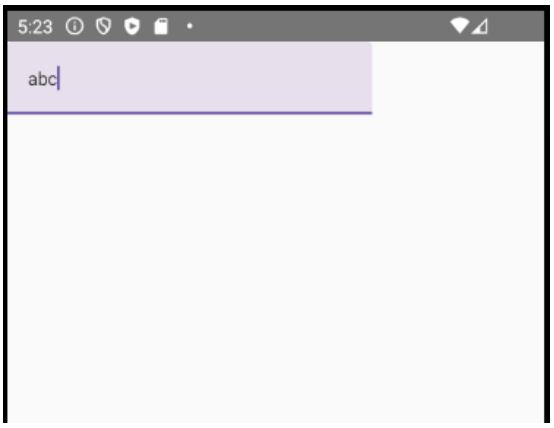
Update MainScreen – Use rememberSaveable and mutableStateOf

Update MainScreen so that the name variable uses both rememberSaveable (instead of remember) and mutableStateOf in its declaration. Change it to the following:

```
var name by rememberSaveable { mutableStateOf("") }
```

Run the App

Type in the TextField and rotate the device. The data that was typed in before the rotation should still be there. Here are screenshots of what it should like before and after the rotation:



Hoist state up to another function

The TextField's state is currentlly in the same composable function that it was defined in. We will now update the app so that the TextField's state is hoisted up to the calling function (the function that calls the function where TextField is defined).

Create a Kotlin file named MyReusableTextField.kt. Add a composable function named MyReusableTextField. This function is now going to create the TextField. Here is the function header:

```
@Composable
fun MyReusableTextField(name: String, onNameChange: (String)->Unit, modifier: Modifier = Modifier)
```

The screen should look the same as before (only one TextField).

More specifications:

- MyReusableTextField
    - Add a TextField
        - Set the TextField's value to the name parameter (we do not need a local variable since we are using the parameter instead).
        - Set the TextField's onValueChange to onNameChange (replace {name = it} with onNameChange).
        - Set the TextField's modifier to the modifier parameter of MainScreenReusable.
- Update MainScreen as follows:
    - Remove the TextField.
    - Add a call to MyReusableTextField. You should pass in the following parameters:
        - A value for the name variable
        - An appropriate function reference for onNameChange (use the same function body as was previously used by onValueChange).
        - The modifier parameter.

Run the App

Everything should still work the same as before you made the changes.