**Lab Exercise 14- State Hoisting in Jetpack Compose**

**Objective:**

In this lab, you will learn the differences between managing state within composables and hoisting state to a higher level, such as the MainActivity. We will create two composables: one for displaying a counter and another for controlling it, and we will hoist the state to the MainActivity to observe how state management works at different levels.

**Steps:**

**Step 1: Set Up a New Jetpack Compose Project**

1. Open **Android Studio**.
2. Create a new project:
   * Select **New Project** -> **Empty Compose Activity**.
   * Set the project name to StateHoistingActivityDemo.
3. Ensure your project includes the necessary Jetpack Compose dependencies in build.gradle:

**Step 2: Create Two Composables Without State Hoisting**

We will first create two composables (CounterDisplay and CounterButtons) where each manages its own state internally.

1. Open MainActivity.kt and start by defining two composables with internal state.

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.layout.\*

import androidx.compose.material3.\*

import androidx.compose.runtime.\*

import androidx.compose.ui.Alignment

import androidx.compose.ui.Modifier

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import com.example.statehoistingactivitydemo.ui.theme.StateHoistingActivityDemoTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

StateHoistingActivityDemoTheme {

NoHoistingCounterApp() // Call composables without state hoisting

}

}

}

}

// Composable managing its own internal state

@Composable

fun CounterDisplay() {

var count by remember { mutableStateOf(0) }

Column(

modifier = Modifier

.fillMaxSize()

.padding(16.dp),

verticalArrangement = Arrangement.Center,

horizontalAlignment = Alignment.CenterHorizontally

) {

Text(text = "Counter: $count", style = MaterialTheme.typography.headlineLarge)

Spacer(modifier = Modifier.height(16.dp))

Row {

Button(onClick = { count++ }, modifier = Modifier.padding(end = 8.dp)) {

Text(text = "Increment")

}

Button(onClick = { if (count > 0) count-- }) {

Text(text = "Decrement")

}

}

}

}

@Preview(showBackground = true)

@Composable

fun CounterDisplayPreview() {

StateHoistingActivityDemoTheme {

CounterDisplay()

}

}

**Explanation:**

* The CounterDisplay composable manages its own state (count).
* The UI includes a counter value and buttons to increment or decrement it.

**Step 3: Refactor to Use State Hoisting in the Parent Composable**

Next, you will refactor the code to use **State Hoisting** and separate the UI logic into two composables: one to display the counter and another to control it. The state will be hoisted to a parent composable (NoHoistingCounterApp).

1. Modify the MainActivity.kt as follows to hoist the state in the parent composable.

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

StateHoistingActivityDemoTheme {

HoistingCounterApp() // Call composables with state hoisting

}

}

}

}

// Parent composable managing the state and passing it to children

@Composable

fun HoistingCounterApp() {

var count by remember { mutableStateOf(0) }

Column(

modifier = Modifier

.fillMaxSize()

.padding(16.dp),

verticalArrangement = Arrangement.Center,

horizontalAlignment = Alignment.CenterHorizontally

) {

// Pass the count state to the display composable

CounterDisplay(count)

Spacer(modifier = Modifier.height(16.dp))

// Pass the update functions to the button composable

CounterButtons(

onIncrement = { count++ },

onDecrement = { if (count > 0) count-- }

)

}

}

// Composable that only displays the counter, receives count as parameter

@Composable

fun CounterDisplay(count: Int) {

Text(text = "Counter: $count", style = MaterialTheme.typography.headlineLarge)

}

// Composable that has the buttons to modify the counter

@Composable

fun CounterButtons(onIncrement: () -> Unit, onDecrement: () -> Unit) {

Row {

Button(onClick = onIncrement, modifier = Modifier.padding(end = 8.dp)) {

Text(text = "Increment")

}

Button(onClick = onDecrement) {

Text(text = "Decrement")

}

}

}

@Preview(showBackground = true)

@Composable

fun HoistingCounterAppPreview() {

StateHoistingActivityDemoTheme {

HoistingCounterApp()

}

}

**Explanation:**

* The **state is hoisted** to the HoistingCounterApp composable.
* CounterDisplay receives the count as a parameter, while CounterButtons receives the onIncrement and onDecrement event handlers.

**Step 4: Hoist the State to the MainActivity**

Now, let's hoist the state all the way up to the MainActivity. This approach shows how state management can move out of the composables and be controlled at the activity level.

1. Modify the MainActivity.kt to manage the state inside MainActivity.

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

// Hoisting the state to the Activity level

var count by mutableStateOf(0)

setContent {

StateHoistingActivityDemoTheme {

Column(

modifier = Modifier

.fillMaxSize()

.padding(16.dp),

verticalArrangement = Arrangement.Center,

horizontalAlignment = Alignment.CenterHorizontally

) {

// Passing the state to composables from the Activity

CounterDisplay(count)

Spacer(modifier = Modifier.height(16.dp))

CounterButtons(

onIncrement = { count++ },

onDecrement = { if (count > 0) count-- }

)

}

}

}

}

}

**Explanation:**

* The state (count) is now managed directly inside the MainActivity itself.
* The composables are still stateless, receiving the state and event handlers as parameters from the MainActivity.

**Step 5: Run the Application**

1. Select an emulator or a physical device.
2. Run the project.
3. You should observe:
   * In the **NoHoistingCounterApp**, the state is managed locally inside the composable (CounterDisplay).
   * In the **HoistingCounterApp**, the state is hoisted to the parent composable, making the child composables stateless.
   * Finally, when state is hoisted to the MainActivity, state management occurs at the activity level, and composables are fully stateless.