ViewModel dan State dalam Compose

ViewModel dan **State** adalah dua konsep yang sangat penting dalam pengembangan aplikasi dengan Jetpack Compose karena membantu mengelola data dan status UI dengan cara yang efisien.

ViewModel

ViewModel adalah komponen arsitektur aplikasi Android yang direkomendasikan untuk menyimpan dan mengelola data terkait UI yang sesuai dengan siklus proses aplikasi. ViewModel dirancang untuk menyimpan dan mengelola data terkait UI dengan cara yang terlepas dari siklus proses. Ini berarti bahwa data ViewModel tidak hilang saat aktivitas atau fragmen dihancurkan dan dibuat kembali karena perubahan konfigurasi, seperti rotasi layar.

ViewModel adalah bagian dari Android Architecture Components yang dirancang untuk menyimpan dan mengelola data yang terkait dengan UI dalam siklus hidupnya. Tujuan utama ViewModel adalah agar data tetap ada saat terjadi perubahan siklus hidup seperti rotasi layar.

Fungsi utama ViewModel:

- Menyediakan data ke UI.
- Memisahkan logika aplikasi dari logika tampilan.
- Mencegah kehilangan data saat aktivitas atau fragment di-recreate.

Dalam Jetpack Compose, ViewModel biasanya digunakan bersama dengan **State** (status) untuk mempermudah pengelolaan status UI.

Cara menggunakan ViewModel:

1. Buat kelas ViewModel.
2. Simpan data yang ingin dikelola sebagai `MutableState`.
3. Hubungkan ViewModel dengan UI menggunakan `viewModel()`.

```
class MyViewModel : ViewModel() {
    // Menggunakan state untuk data UI
    var count by mutableStateOf(0)
        private set

    fun incrementCount() {
        count++
    }
}
```

Menggunakan ViewModel di Compose:

```
@Composable
fun MyScreen(viewModel: MyViewModel = viewModel()) {
    val count = viewModel.count
```

```
    Column {
        Text(text = "Count: $count")
        Button(onClick = { viewModel.incrementCount() }) {
            Text("Increment")
        }
    }
}
```

State dalam Jetpack Compose:

Dalam konteks Compose, status mengacu pada data apa pun yang dapat memengaruhi UI. Ini bisa berupa apa saja mulai dari nilai primitif seperti Int atau String hingga objek yang lebih kompleks. Saat status berubah, Compose secara otomatis memperbarui UI untuk mencerminkan perubahan tersebut.

State adalah inti dari Jetpack Compose yang memungkinkan UI untuk bereaksi terhadap perubahan data secara otomatis. Konsep utama Compose adalah "UI = f(State)", artinya UI akan otomatis berubah berdasarkan state yang diperbarui.

Cara kerja State:

- Ketika state berubah, Compose akan merender ulang bagian UI yang menggunakan state tersebut.
- Jetpack Compose memiliki remember dan mutableStateOf untuk menyimpan dan mengelola state di dalam composable.

Contoh sederhana penggunaan State:

```
@Composable
fun CounterScreen() {
    var count by remember { mutableStateOf(0) }

    Column {
        Text(text = "Count: $count")
        Button(onClick = { count++ }) {
            Text("Increment")
        }
    }
}
```

Hubungan ViewModel dan State di Compose:

ViewModel dan status sering digunakan bersama dalam aplikasi Compose. ViewModel bertanggung jawab untuk menyimpan dan mengelola status, sementara Compose bertanggung jawab untuk menampilkan status tersebut di UI dan memperbaruinya saat berubah.

1. ViewModel menyimpan state data secara global untuk UI.
2. State digunakan di dalam composable untuk menangani status UI secara lokal.

3. Dengan ViewModel, kita bisa mengelola state yang tetap ada di luar siklus hidup composable.

Contoh gabungan ViewModel dan State:

```
class MyViewModel : ViewModel() {
    var count by mutableStateOf(0)
        private set

    fun increment() {
        count++
    }
}

@Composable
fun MyAppScreen(viewModel: MyViewModel = viewModel()) {
    val count = viewModel.count

    Column {
        Text(text = "Count: $count")
        Button(onClick = { viewModel.increment() }) {
            Text("Increment")
        }
    }
}
```

Perbedaan State Lokal vs State di ViewModel:

State Lokal (remember, mutableStateOf)	State di ViewModel
Hidup hanya selama composable ada.	Tetap ada selama aktivitas/fragment hidup.
Cocok untuk UI yang sederhana dan transient.	Cocok untuk data yang harus persist.
Tidak mendukung rotasi atau rekreasi ulang.	Mendukung rotasi dan rekreasi ulang.

Berikut adalah contoh sederhana project yang menggunakan **ViewModel** dan **State** dalam aplikasi Android berbasis **Jetpack Compose**. Kita akan membuat aplikasi penghitung (*Counter App*) di mana data dikelola oleh **ViewModel**, dan status UI diperbarui menggunakan **State**.

Struktur Proyek

1. Buat **ViewModel** untuk mengelola data.
2. Buat UI dengan **State** yang terhubung ke **ViewModel**.
3. Gunakan viewModel() untuk menghubungkan ViewModel ke composable.

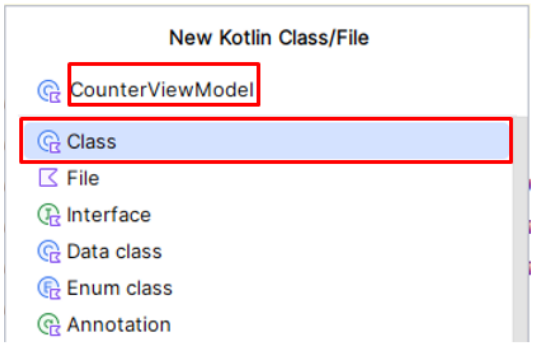
Langkah 1: Tambahkan Dependensi

Pastikan project Anda sudah mendukung Jetpack Compose dan ViewModel. Tambahkan dependensi berikut di file `build.gradle.kts(Module : app)`

```
dependencies {
    . . .
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")
    . . .
}
```

Langkah 2: Buat ViewModel

Buat file baru `CounterViewModel.kt`: Pada package com.example.counterapp, klik kanan pilih menu **New > Kotlin Class/File**. Pilih opsi **Class**, dan isikan `CounterViewModel`, tekan [enter]

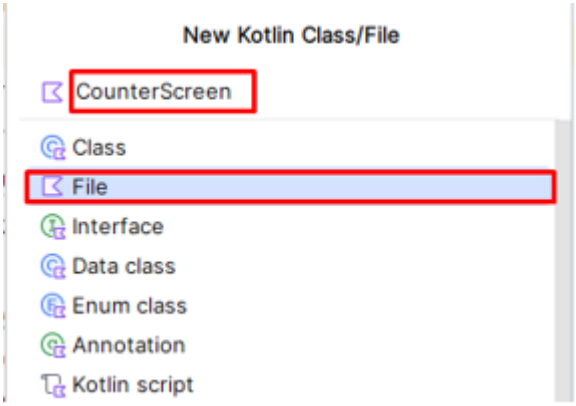


Tulis kode seperti berikut:

```
package com.example.counterapp

import androidx.lifecycle.ViewModel
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue

class CounterViewModel : ViewModel() {
    // State untuk menyimpan nilai count
    var count by mutableStateOf(0)
        private set

    // Fungsi untuk meningkatkan count
    fun increment() {
        count++
    }

    // Fungsi untuk mengurangi count
    fun decrement() {
        if (count > 0) count--
    }
}
```

Langkah 3: Buat UI dengan Compose

Buat file baru `CounterScreen.kt`: Pada package com.example.counterapp, klik kanan pilih menu **New > Kotlin Class/File**. Pilih opsi **File**, dan isikan `CounterScreen`, tekan [enter].



Tulis kode seperti berikut:

```
package com.example.counterapp

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.lifecycle.viewmodel.compose.viewModel

@Composable
fun CounterScreen(viewModel: CounterViewModel = viewModel()) {
    // Ambil nilai count dari ViewModel
    val count = viewModel.count

    // UI
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = "Count: $count",
```

```
            style = MaterialTheme.typography.headlineMedium
        )
        Row {
            Button(
                onClick = { viewModel.decrement() },
                modifier = Modifier.padding(8.dp)
            ) {
                Text(text = "Decrement")
            }
            Button(
                onClick = { viewModel.increment() },
                modifier = Modifier.padding(8.dp)
            ) {
                Text(text = "Increment")
            }
        }
    }
}
```

Langkah 4: Panggil di MainActivity

Buka file `MainActivity.kt` dan gunakan composable yang sudah dibuat:

```
package com.example.counterapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import com.example.counterapp.ui.theme.CounterAppTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            CounterAppTheme {
                // Aplikasi dimulai di sini
                Surface(color = MaterialTheme.colorScheme.background) {
                    CounterScreen()
                }
            }
        }
    }
}
```

Penjelasan Singkat

1. **ViewModel**:
    - Mengelola status aplikasi (variabel count).
    - Menyediakan fungsi untuk memperbarui data (increment() dan decrement()).
2. **Composable**:
    - UI di CounterScreen membaca data dari ViewModel.
    - Ketika tombol ditekan, data di ViewModel diperbarui, dan UI otomatis dirender ulang.
3. **State dan ViewModel**:
    - var count by mutableStateOf(0) di ViewModel digunakan untuk memastikan state UI tetap sinkron dengan logika bisnis.

Jika Anda jalankan, aplikasi ini memungkinkan pengguna menambah atau mengurangi angka, dan semua data dikelola dengan rapi oleh ViewModel!