BCS 371 Lab – ViewModel

Overview

Create an app that uses a ViewModel to retain UI state data through configuration changes. Use mutableStateOf to display data updates in the UI automatically.

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.

Add ViewModel Dependency

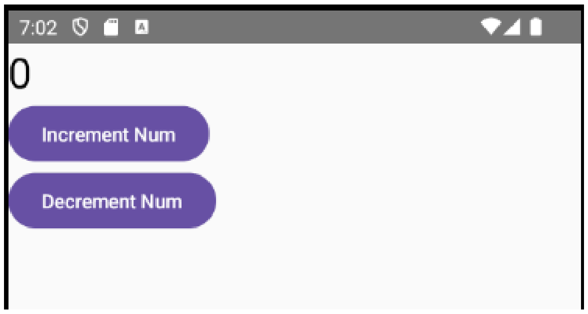
Add the necessary dependencies to allow using the ViewModel architecture component. The dependency should be added to the build.gradle (App) file.

Setup the Main Screen (No View Model)

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

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

It should look like the following:



More specifications for this composable function:

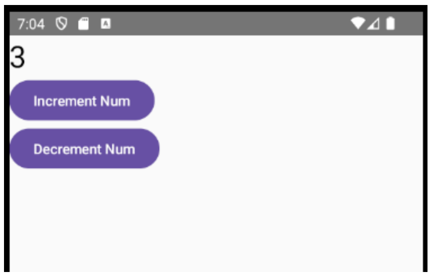
- Declare the following variable at the top of the function:
  var num by remember { mutableStateOf(0) }
- Associate the Text composable with the num variable.
- When the Increment Num button is clicked it should increment the num variable.
- When the Decrement Num button is clicked it should decrement the num variable.
- Update MainActivity to call MainScreenNoViewModel().

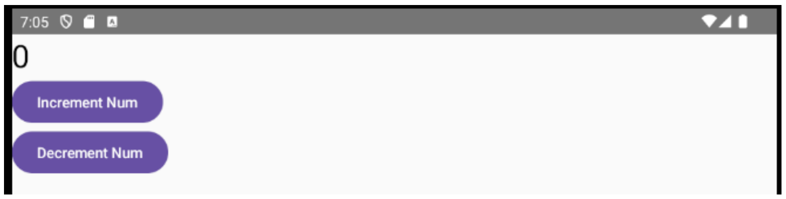
Run the App

You should see the main screen appear when you run the app.

- Press the increment and decrement buttons a few times each to update the num value.
- Make sure that Auto-rotate is set to on for the emulator.
- Now rotate the device using the Rotate left button on the emulator (the Rotate left button is in the display shown to the right of the main emulator screen). The num value will be reset to 0. This happens because the local variable is not retained through the configuration change caused by the left rotation.

Here are screenshots before and after rotating the device:





Create MainScreenViewModel Class

Add a new ViewModel subclass named MainScreenViewModel.

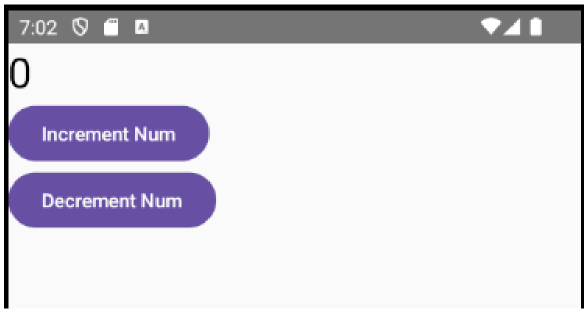
- It should contain one member variable named num that is initialized to 0. Put the member variable inside the { } of the class definition (not inside a primary constructor). You can use the following code to declare the variable:
  var num by mutableStateOf(0)
  Note: We are using mutableStateOf because we want it to send out notifications when its value changes.
- Add a function to increment num by 1.
- Add a function to decrement num by 1.

Setup the Main Screen (View Model)

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

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

It should look like the following (same display as before):



More specifications for this composable function:

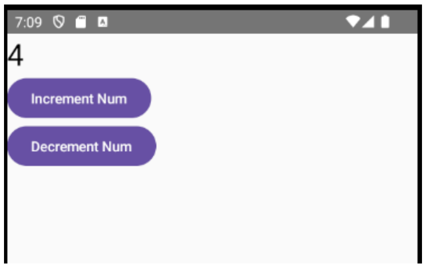
- Get an instance of MainScreenViewModel and store in a local variable:
  val viewModel = viewModel { MainScreenViewModel() }
- Associate the Text composable with the num variable on the view model.
- When the Increment Num button is clicked call the function on the view model to increment the num variable.
- When the Decrement Num button is clicked call the function on the view model to decrement the num variable.
- Update MainActivity to call MainScreenWithViewModel() (instead of MainScreenNoViewModel()).

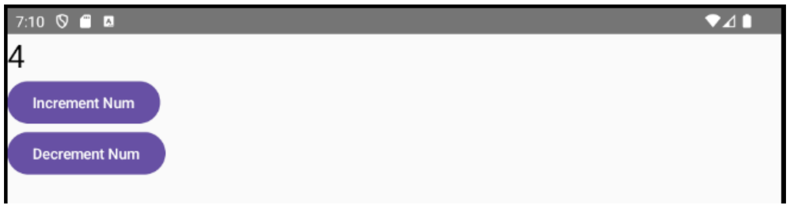
Run the App

You should see the main screen appear when you run the app.

- Press the increment and decrement buttons a few times each to update the num value.
- Make sure that Auto-rotate is set to on for the emulator.
- Now rotate the device using the Rotate left button on the emulator (the Rotate left button is in the display shown to the right of the main emulator screen). The value should now be retained through the configuration change caused by the left rotation.

Here are screenshots before and after rotating the device:





Expose Only Immutable State

Only expose immutable state in properties on the view model class. Make it so that the mutableStateOf variable can only be updated from inside the view model class. The app should still operate the same from the user perspective.