CS 426 – Mobile Application Development

Week 6 Lab : Advanced Navigation & Gestures with Fragments

Instructor : Dr. Slim Namouchi
Date : Saturday 25 Oct 2025

Objectives :

- Extend the Week 5 Calculator App (Fragments + Safe Args) with gesture navigation.
- Implement a Shared ViewModel to manage app state across fragments.
- Build a multi-view architecture using the Android Navigation Component.
- Understand gesture detection, touch listeners, and communication principles between fragments.
- Apply Clean Architecture and MVVM patterns in a real-world context.

Lab Overview

We will evolve our *Calculator App* into an interactive, modern Android application by introducing:

Feature	Description	Benefit
Swipe Gestures	Left / Right gesture-based navigation	Smooth UX, natural interaction
Shared ViewModel	Central data store shared across fragments	Persistent state across navigation
LiveData	Observed data that updates UI automatically	Reactive UI, cleaner architecture
ViewBinding	Type-safe UI element access	Prevents runtime crashes

This lab moves students from simple fragment communication to an MVVM-based navigation architecture, the same model used in real professional Android apps.

Project Setup

1 Re-use your Week 5 Project

Open your *Calculator App* from Week 5 (3 Fragments + Safe Args + Navigation).
If you prefer to start fresh, duplicate it and rename the folder :

CalculatorApp_Week6_Gestures

2 Gradle Dependencies

Open app/build.gradle and confirm the following sections exist :

dependencies {

// Lifecycle ViewModel + LiveData

```
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")
implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.6.2")
implementation("androidx.fragment:fragment-ktx:1.6.2")
```

Then → **Sync Project with Gradle Files.**

3 Recommended Folder Structure

```
app/
 ├─ java/com/example/calculator/
 │   ├─ MainActivity.kt
 │   ├─ fragments/
 │   │   ├─ WelcomeFragment.kt
 │   │   ├─ CalculatorFragment.kt
 │   │   └─ ResultFragment.kt
 │   └─ viewmodel/
 │       └─ SharedCalcViewModel.kt
 └─ res/
     ├─ layout/
     │   ├─ fragment_welcome.xml
     │   ├─ fragment_calculator.xml
     │   └─ fragment_result.xml
     └─ navigation/nav_graph.xml
```

4 Clean Architecture Recap

MVVM Pattern (Used Here):

Layer	Responsibility	Example
Model	Holds data & business logic	SharedCalcViewModel
View	Displays UI (XML + Binding)	fragment_calculator.xml
ViewModel	Connects UI and logic	Observes & updates LiveData

🧠 Step 1 – Create SharedCalcViewModel

File : viewmodel/SharedCalcViewModel.kt

```
class SharedCalcViewModel : ViewModel() {

    // Private mutable LiveData
    private val _num1 = MutableLiveData<Double>()
    private val _num2 = MutableLiveData<Double>()
    private val _operation = MutableLiveData<String>()
    private val _result = MutableLiveData<Double>()

    // Public immutable accessors
    val num1: LiveData<Double> get() = _num1
    val num2: LiveData<Double> get() = _num2
    val operation: LiveData<String> get() = _operation
    val result: LiveData<Double> get() = _result

    /**
     * Perform calculation and update LiveData
     */
    fun calculate(num1: Double, num2: Double, op: String) {
        _num1.value = num1
        _num2.value = num2
        _operation.value = op

        val calculatedResult = when (op) {
            "+" -> num1 + num2
            "-" -> num1 - num2
            "*" -> num1 * num2
            "/" -> if (num2 != 0.0) num1 / num2 else 0.0
            else -> 0.0
        }

        _result.value = calculatedResult
    }

    /**
     * Clear all data
     */
    fun clear() {
        _num1.value = 0.0
        _num2.value = 0.0
        _operation.value = ""
```

```
            _result.value = 0.0
        }
    }
```

Explanation :

This ViewModel stores the calculator's state.

Because it's scoped to the **activity**, all fragments share the same instance — values survive fragment changes and configuration rotations.

💬 Step 2 – Use the Shared ViewModel in CalculatorFragment

File : CalculatorFragment.kt

```
class CalculatorFragment : Fragment() {

    private lateinit var binding: FragmentCalculatorBinding
    private val viewModel: SharedCalcViewModel by activityViewModels()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentCalculatorBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.btnAdd.setOnClickListener { handleOperation("+") }
        binding.btnSub.setOnClickListener { handleOperation("-") }
        binding.btnMul.setOnClickListener { handleOperation("*") }
        binding.btnDiv.setOnClickListener { handleOperation("/") }

        binding.btnClear.setOnClickListener {
            binding.etNum1.text?.clear()
            binding.etNum2.text?.clear()
        }
    }

    private fun handleOperation(op: String) {
        val num1 = binding.etNum1.text.toString().toDoubleOrNull()
        val num2 = binding.etNum2.text.toString().toDoubleOrNull()

        if (num1 == null || num2 == null) {
            Toast.makeText(requireContext(),
getString(R.string.error_enter_numbers), Toast.LENGTH_SHORT).show()
            return
        }

        if (op == "/" && num2 == 0.0) {
            Toast.makeText(requireContext(),
getString(R.string.error_divide_zero), Toast.LENGTH_SHORT).show()
```

```
            return
        }

        // Store data in ViewModel
        viewModel.calculate(num1, num2, op)

        // Navigate to ResultFragment
        findNavController().navigate(
            R.id.action_calculatorFragment2_to_resultFragment2
        )
    }
}
```

Notes :

- activityViewModels() ensures one shared ViewModel.
- Input validation prevents crashes.
- Navigation now depends on the shared state, not on Safe Args.

📊 Step 3 – Observe Data in ResultFragment

```
class ResultFragment : Fragment() {

    private lateinit var binding: FragmentResultBinding
    private val viewModel: SharedCalcViewModel by activityViewModels()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentResultBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Observe LiveData from ViewModel
        viewModel.num1.observe(viewLifecycleOwner) { num1 ->
            updateUI()
        }

        viewModel.num2.observe(viewLifecycleOwner) { num2 ->
            updateUI()
        }

        viewModel.operation.observe(viewLifecycleOwner) { operation ->
            updateUI()
        }
```

```
        viewModel.result.observe(viewLifecycleOwner) { result ->
            updateUI()
        }

        binding.btnBack.setOnClickListener {
            findNavController().navigateUp()
        }
    }

    private fun updateUI() {
        val num1 = viewModel.num1.value ?: 0.0
        val num2 = viewModel.num2.value ?: 0.0
        val op = viewModel.operation.value ?: ""
        val result = viewModel.result.value ?: 0.0

        val opName = when (op) {
            "+" -> getString(R.string.addition)
            "-" -> getString(R.string.subtraction)
            "*" -> getString(R.string.multiplication)
            "/" -> getString(R.string.division)
            else -> ""
        }

        binding.tvOperation.text = opName
        binding.tvResult.text = result.toString()
        binding.tvExpression.text = "$num1 $op $num2"
    }
}
```

💡 The fragment listens to LiveData. Whenever the ViewModel updates, the UI automatically reflects the change, no manual refresh required.

🖐️ Step 4 – Add Swipe Gestures for Navigation

Goal: enable a left-swipe on CalculatorFragment to navigate to ResultFragment.

```
private lateinit var gestureDetector: GestureDetector
```

Tip: you can invert the condition for swipe right to go back.

🎨 Step 5 – UI Enhancements

Add these hints in your XML layouts:

fragment_calculator.xml

<TextView

android:id="@+id/tvHint"

android:text="Swipe → to see result"

```
    android:gravity="center"

    android:textStyle="italic"

    android:layout_marginTop="16dp" />
```

fragment_result.xml

Add small motion feedback (optional):

```
val vibrator = requireContext().getSystemService(Context.VIBRATOR_SERVICE) as Vibrator

vibrator.vibrate(VibrationEffect.createOneShot(50, VibrationEffect.DEFAULT_AMPLITUDE))
```