CS 426 – Mobile Application Development

Instructor: Slim NAMOUCHI

Week 5 Lab: Convert Calculator App to Fragments with Safe Args

Lab Objective

This week, you'll transform your multi-activity calculator app into a single-activity app that uses Fragments for each screen: Welcome, Calculator, and Result. You will learn how to use the Android Navigation Component and the Safe Args plugin to navigate between fragments and safely pass data.

Before We Start: Understanding Fragments and Single-Activity Architecture

What is a Fragment?

A **Fragment** in Android represents a portion of your UI inside an activity. Think of it as a modular, reusable UI component with its own lifecycle, layout, and logic but living inside an activity.

Unlike activities, fragments are lightweight and allow you to partition your UI to adapt dynamically to different screen sizes, such as phones and tablets.

Why Use Fragments?

Your previous app uses multiple activities for each screen, which works, but has some downsides:

- Activities are heavyweight; switching between them can be resource-intensive.
- Navigating between activities can be complex and harder to manage back stack.
- Passing data between activities requires intents and careful key management.

Using fragments inside a **single activity** brings benefits:

- Easier and more flexible UI composition (side-by-side fragments on tablets).
- Simplified navigation and back stack management using the Navigation Component.
- Shared data management between fragments in the same activity.

- Reuse UI components across your app.

More information: https://developer.android.com/guide/fragments?hl=fr

Step 1: Prepare Your Project (Single Activity with Multiple Fragments)

Step 1.1: Convert Your App to Single Activity

- Keep only **one** `MainActivity` which will serve as a container for your different screens.
- This `MainActivity` holds a container for swapping fragments dynamically.
- Fragments will represent your app's different screens: Welcome, Calculator, and Result.

Step 1.2: Why Single Activity?

Android architecture guidelines recommend using a single activity with multiple fragments to improve:

- **Performance:** Activities are heavy, and fewer activity transitions improve smoothness.
- **Navigation:** Back stack, deep linking, and animations are easier to control with Navigation Component.
- **Flexibility:** Support for large screens and adaptive layouts.

Step 1.3: Add Navigation Component to Your Project

Modify your Gradle files to include Navigation and Safe Args plugins.

In your project-level build.gradle:

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.5.3"  // Add
this line
        // ... other classpaths ...
    }
}
```

In your app-level build.gradle:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'androidx.navigation.safeargs.kotlin'  // Apply the Safe Args plugin
}

dependencies {
    implementation "androidx.navigation:navigation-fragment-ktx:2.5.3"
    implementation "androidx.navigation:navigation-ui-ktx:2.5.3"
}
```

Step 1.4: Sync the Project

After modifying Gradle files, **sync** your project so all dependencies and plugins are downloaded and configured.

Step 1.5: Setup Your MainActivity Layout

Create or update your `activity_main.xml` to include a **NavHostFragment**, which will host your fragments and handle navigation.

```
<fragment
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:navGraph="@navigation/nav_graph"
    app:defaultNavHost="true"/>
```

Step 1.6: Create Navigation Graph

Inside `res/navigation`, create `nav_graph.xml` where you will define your fragments and navigation paths. We'll cover this in Step 3 in detail.

Step 2: Create Your Navigation Graph (res/navigation/nav_graph.xml)

Define your fragments and the navigation actions between them. Here is a sample:

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/nav_graph"
    app:startDestination="@id/welcomeFragment">

    <fragment
        android:id="@+id/welcomeFragment"
        android:name="com.lostitem.week_5.WelcomeFragment"
        android:label="Welcome">
        <action
            android:id="@+id/action_welcome_to_calculator"
            app:destination="@id/calculatorFragment" />
    </fragment>

    <fragment
        android:id="@+id/calculatorFragment"
        android:name="com.lostitem.week_5.CalculatorFragment"
        android:label="Calculator">
        <action
            android:id="@+id/action_calculator_to_result"
            app:destination="@id/resultFragment">

            <!-- Define Safe Args here -->
            <argument
                android:name="num1"
                app:argType="double" />
            <argument
                android:name="num2"
                app:argType="double" />
            <argument
                android:name="operation"
                app:argType="string" />
            <argument
                android:name="result"
                app:argType="double" />
        </action>
    </fragment>

    <fragment
        android:id="@+id/resultFragment"
        android:name="com.lostitem.week_5.ResultFragment"
        android:label="Result" />

</navigation>
```

Tip: Defining arguments in navigation actions allows Safe Args to generate type-safe classes for you.

Step 3: Create Fragments and Layouts

1. For each screen, create a fragment class and corresponding XML layout:
   - `WelcomeFragment.kt` and `fragment_welcome.xml`
   - `CalculatorFragment.kt` and `fragment_calculator.xml`
   - `ResultFragment.kt` and `fragment_result.xml`
2. Use ViewBinding in fragments to access views safely.
3. Example: `WelcomeFragment.kt`

```
class WelcomeFragment : Fragment() {

    private var _binding: FragmentWelcomeBinding? = null
    private val binding get() = _binding!!

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.btnStart.setOnClickListener {
            findNavController().navigate(R.id.action_welcome_to_calculator)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Tip: `findNavController()` retrieves the navigation controller from the fragment to trigger fragment transactions.

Step 4: Pass Data Safely Using Safe Args

In `CalculatorFragment.kt`, after calculation and validation finish, prepare data and navigate:

```
val action = CalculatorFragmentDirections.actionCalculatorToResult(
    num1 = num1,
    num2 = num2,
    operation = op,
    result = calculationResult
)
findNavController().navigate(action)
```

Tip: Build your project after editing `nav_graph.xml` so Safe Args classes are generated.

Step 5: Receive Data Safely in `ResultFragment.kt`

```
class ResultFragment : Fragment() {

    private var _binding: FragmentResultBinding? = null
    private val binding get() = _binding!!
    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)

        val args = ResultFragmentArgs.fromBundle(requireArguments())

        binding.tvExpression.text = "${args.num1} ${args.operation} ${args.num2}"
        binding.tvResult.text = args.result.toString()
        binding.tvOperation.text = getString(when(args.operation) {
            "+" -> R.string.addition
            "-" -> R.string.subtraction
            "*" -> R.string.multiplication
            "/" -> R.string.division
            else -> R.string.result_label
        })

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

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Tip: Safe Args protects you from key mismatches and type errors.

Step 6: Testing and Validation

- Test navigation from Welcome to Calculator screen.
- Enter invalid inputs to verify validation and error handling.
- Perform operations and verify Result screen correctness.
- Test back button behavior to ensure it correctly returns to previous screen.

Summary Tips

- **Always build your project** after updating nav graph to generate Safe Args classes.
- Use `findNavController()` in fragments for navigation.
- Keep UI logic inside fragments, and use ViewBinding for view access.
- Define navigation arguments to enforce type safety when passing data.
- Test thoroughly including edge cases like division by zero.