Activity Lifecycle & Navigation

Asst. Prof. Monlica Wattana, Ph.D
Department of Computer Science,
Khon Kaen University

SC 362 007 Mobile Device Programming
CP410804 Programming for Mobile Application

Outline

- Activity Lifecycle
- Intents
- Jetpack Navigation

Android Lifecycle

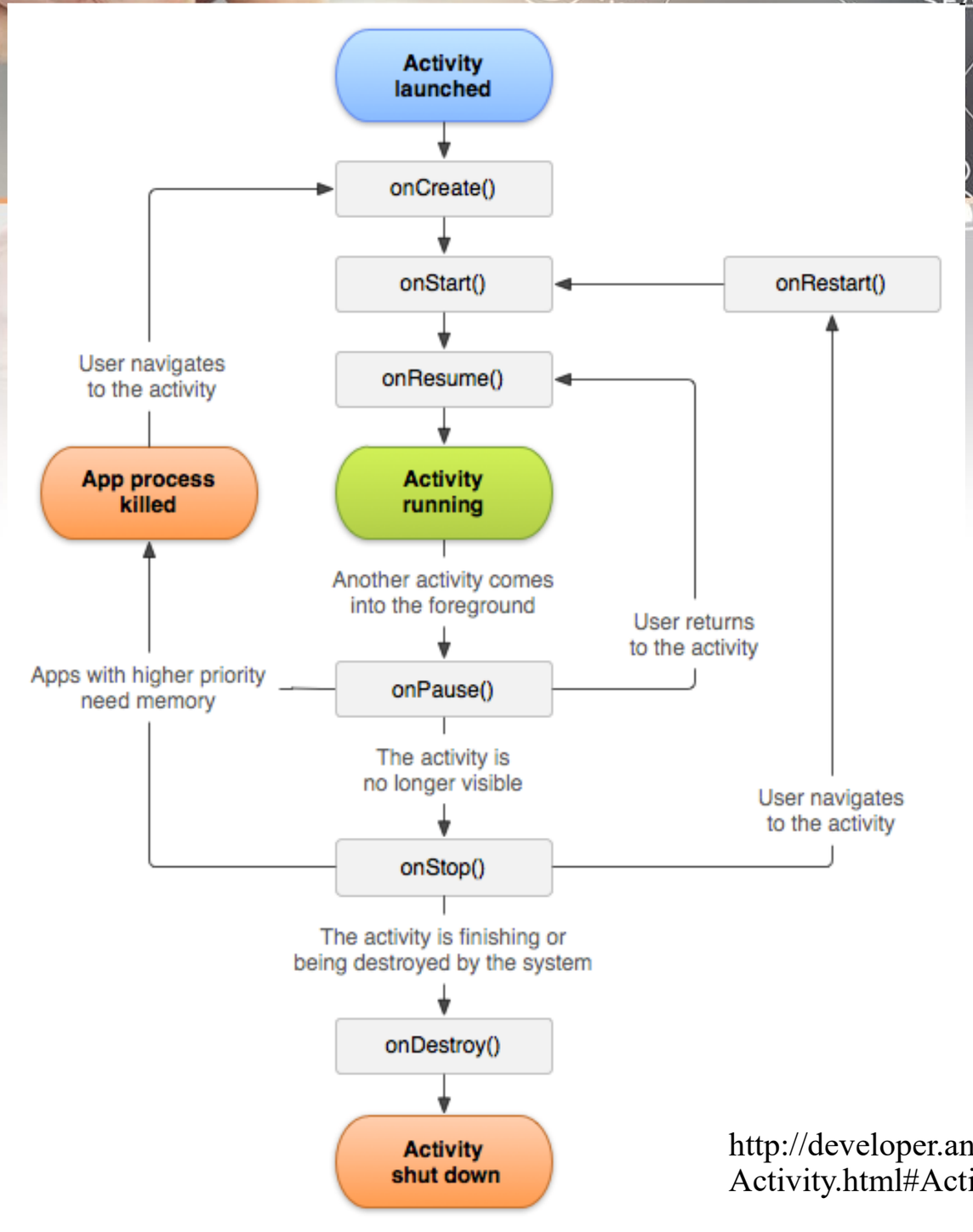
- Each application runs in its own process.
- Each activity of an app is run in the apps process
- Processes are started and stopped as needed to run an apps components.
- Processes may be killed to reclaim needed resources.
- Killed apps may be restored to their last state when requested by the user
- The steps that an application goes through from starting to finishing

Activity state

- An activity can be thought of as being in one of several states:
  - starting: In the process of loading up, but not fully loaded.
  - running: Done loading and now visible on the screen.
  - paused: Partially obscured or out of focus, but not shut down.
  - stopped: No longer active, but still in the device's active memory.
  - destroyed: Shut down and no longer currently loaded in memory.

Activity Lifecycle

- Transitions between these states are represented by events that programmer can listen to in their activity code.
- The activity class has the following method **callbacks** to help programmer manage the app:
  - onCreate()
  - onStart()
  - onResume()
  - onPause()
  - onStop()
  - onRestart()
  - onDestroy()



http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

Activity Lifecycle

onCreate method

In onCreate, to create and set up the activity object, load any static resources like images, layouts, set up menus etc.

- After this, the Activity object exists
- think of this as the "constructor" of the activity

```
class MainActivity : AppCompatActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState) // always call super
        enableEdgeToEdge()
        setContent { ……} // set up layout
                any other initialization code
            }
        }
    } }
```

Activity Lifecycle

onPause method

- When onPause is called, the activity is still partially visible.
- May be temporary, or on way to termination.
  - Stop animations or other actions that consume CPU.
  - Commit unsaved changes (e.g. draft email).
  - Release system resources that affect battery life.

```
override fun onPause() {
        super.onPause() // always call super
        print("onPause")
    }
}
```

Activity Lifecycle

onResume method

-When onResume is called, your activity is coming out of the Paused state and into the Running state again.

- Also called when activity is first created/loaded!

- **Initialize resources** that you will release in onPause.
- **Start/resume animations** or other ongoing actions that should only run when activity is visible on screen.

```
override fun onResume() {
        super.onResume() // always call super
        if (myConnection == null) {
                myConnection = new ExampleConnect() // init.resources
                myConnection.connect();
        }
}
```

Activity Lifecycle

onStop method

-When onStop is called, your activity is no longer visible on the screen:

- User chose another app from Recent Apps window.
- User starts a different activity in your app.
- User receives a phone call while in your app.

- App might still be running, but that activity is not.

- onPause is always called before onStop.
- onStop performs heavy-duty shutdown tasks like writing to a database.

```
override fun onStop() {
        super.onStop()   // always call super
            ...
}
```

Activity Lifecycle

onStart and onRestart

- onStart is called every time the activity begins.

- onRestart is called when activity was stopped but is started again later (all but the first start).

  - Not as commonly used; favor onResume.
  - Re-open any resources that onStop closed.

```
override fun onStart() {
        super.onStart(); // always call super
        ...
}
```

```
override fun onRestart() {
        super.onRestart(); // always call super
        ...
}
```

Activity Lifecycle

onDestroy method

-When onDestroy is called, your entire app is being shut down and unloaded from memory.

- Unpredictable exactly when/if it will be called.
- Can be called whenever the system wants to reclaim the memory used by your app.
- Generally, favor onPause or onStop because they are called in a predictable and timely manner.

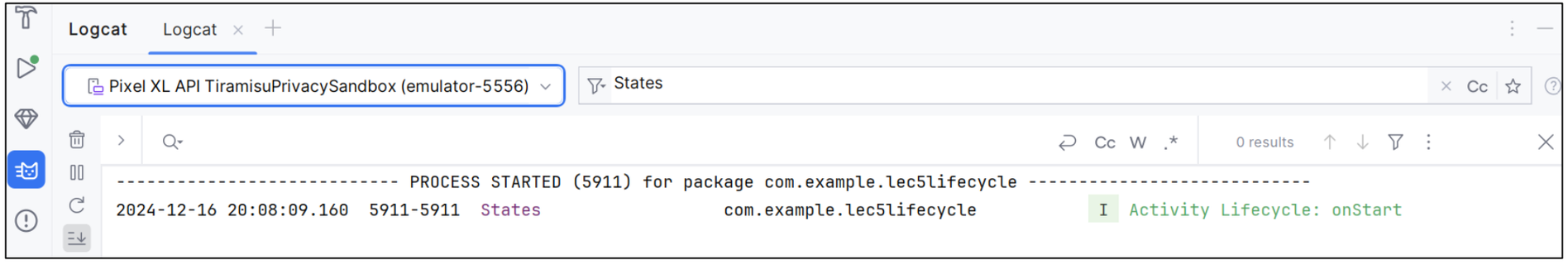
```
override fun onDestroy() {
        super.onDestroy(); // always call super
        ...
}
```

Testing activity states

Use the LogCat system for logging messages when your app changes states:

- analogous to System.out.println debugging for Android apps
- appears in the LogCat console in Android Studio

```
override fun onStart() {
        super.onStart()
        Log.i("States", "Activity Lifecycle: onStart")
}
```



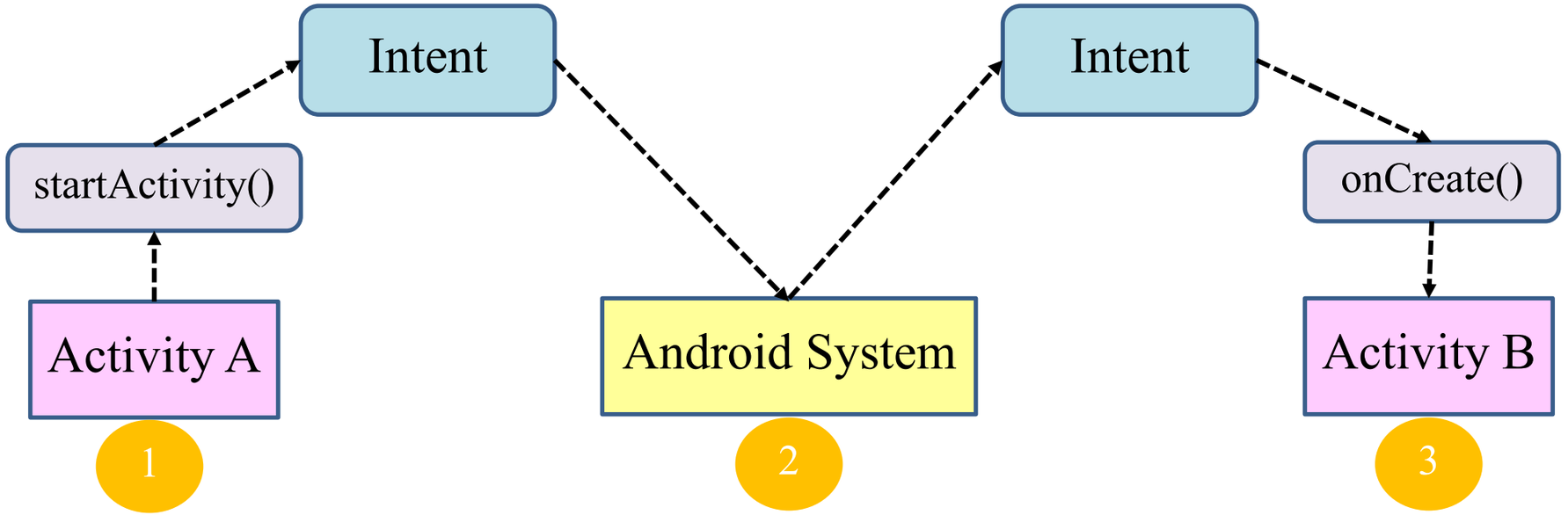
Navigation between screens

There are two primary ways to navigate between screens:

1. Intents: A classic Android navigation method is intents, which is usually used to start external apps or activities. While not the main focus in Compose,
2. Jetpack Navigation: Jetpack Navigation is the recommended approach for building multi-screen apps in Compose.

Intents

- When a new Activity is started, an **Intent** object is created and passed to that Activity
- The Intent object contains information about what the Activity is meant to do, and any data it needs in order to do it



Intents

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Intents

Intents type: there are 2 type

1. Explicit intent
2. Implicit intent

finish() is used to destroy an activity and remove it from the stack.

Explicit intent

Intent statement:

```
val context = LocalContext.current
```

```
val i = Intent(context, OtherActivity::class.java)
context.startActivity(i)
```

An explicit intent tells Android system to run specific component such as ActivityB

```
val i = Intent(applicationContext, ActivityB::class.java)
context.startActivity(i)
```

Implicit intent

- Implicit intents specify the action which should be performed and optionally data which provides content for the action.
- If an implicit intent is sent to the Android system, it searches for all components which are registered for the specific action and the fitting data type.
- If only one component is found, Android starts this component directly.
- If several components are identified by the Android system, the user will get a selection dialog and can decide which component should be used for the intent

Implicit intent

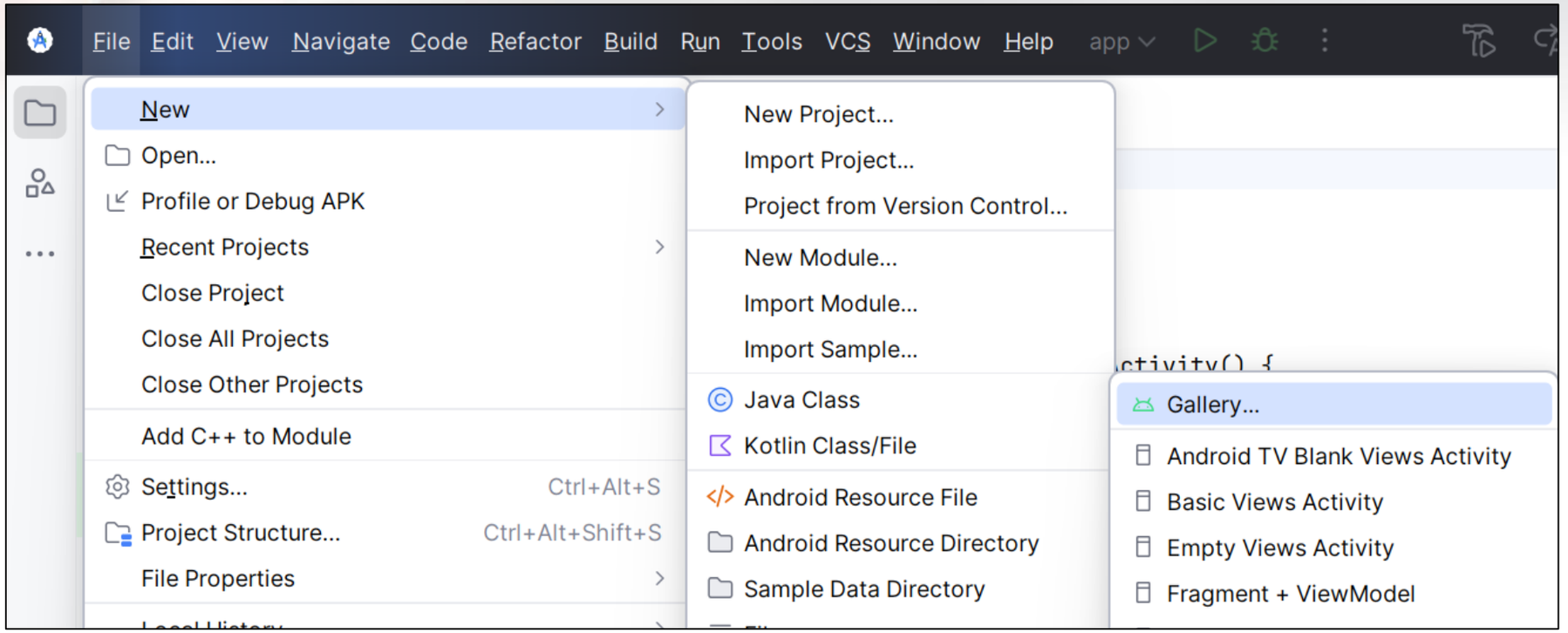
- For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

```
val context = LocalContext.current
```

```
val i = Intent(Intent.ACTION_VIEW)
    i.data = Uri.parse("http://www.google.com")
    context.startActivity(i)
```

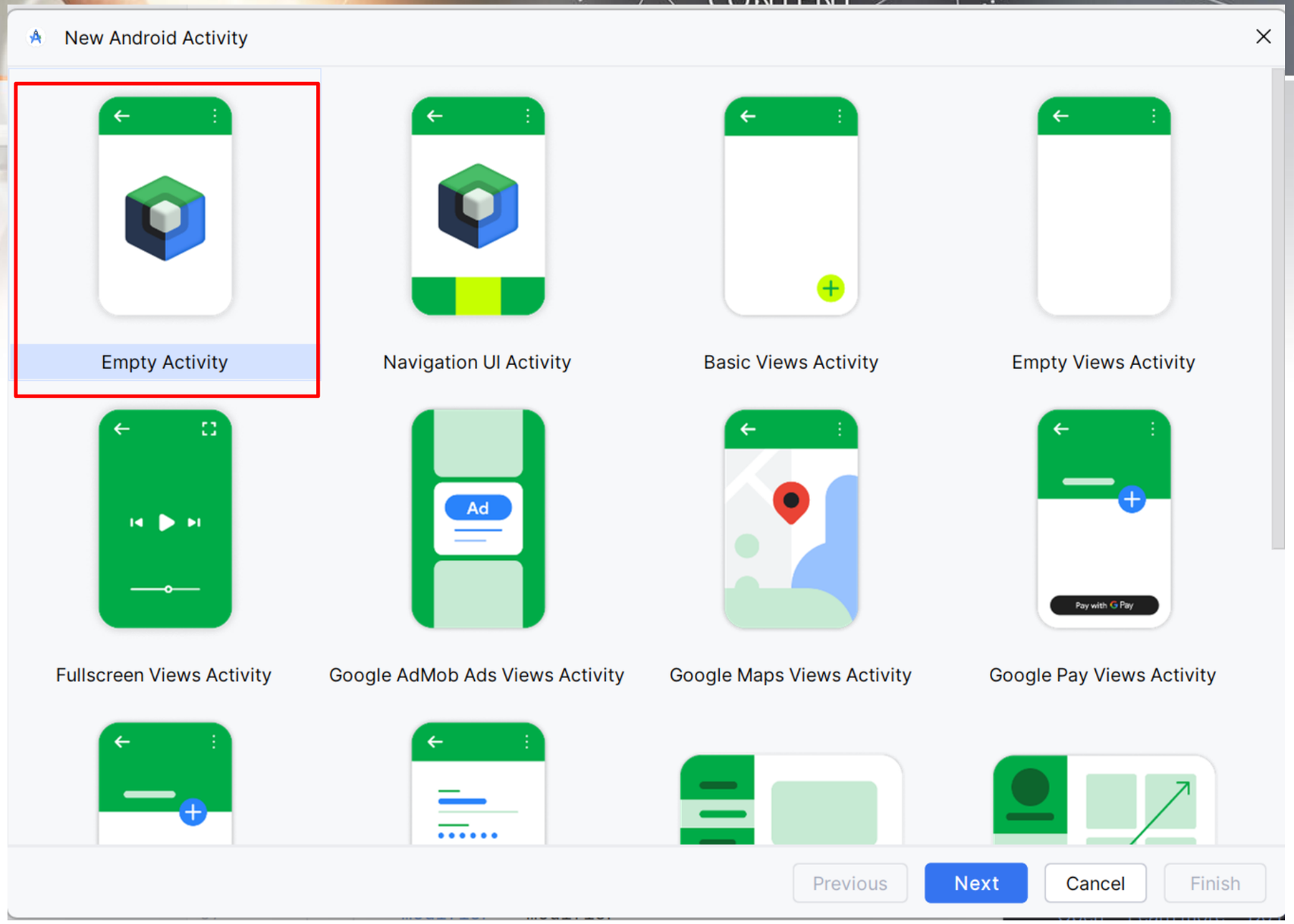
Intent Example

Add a new Activity: SecondActivity



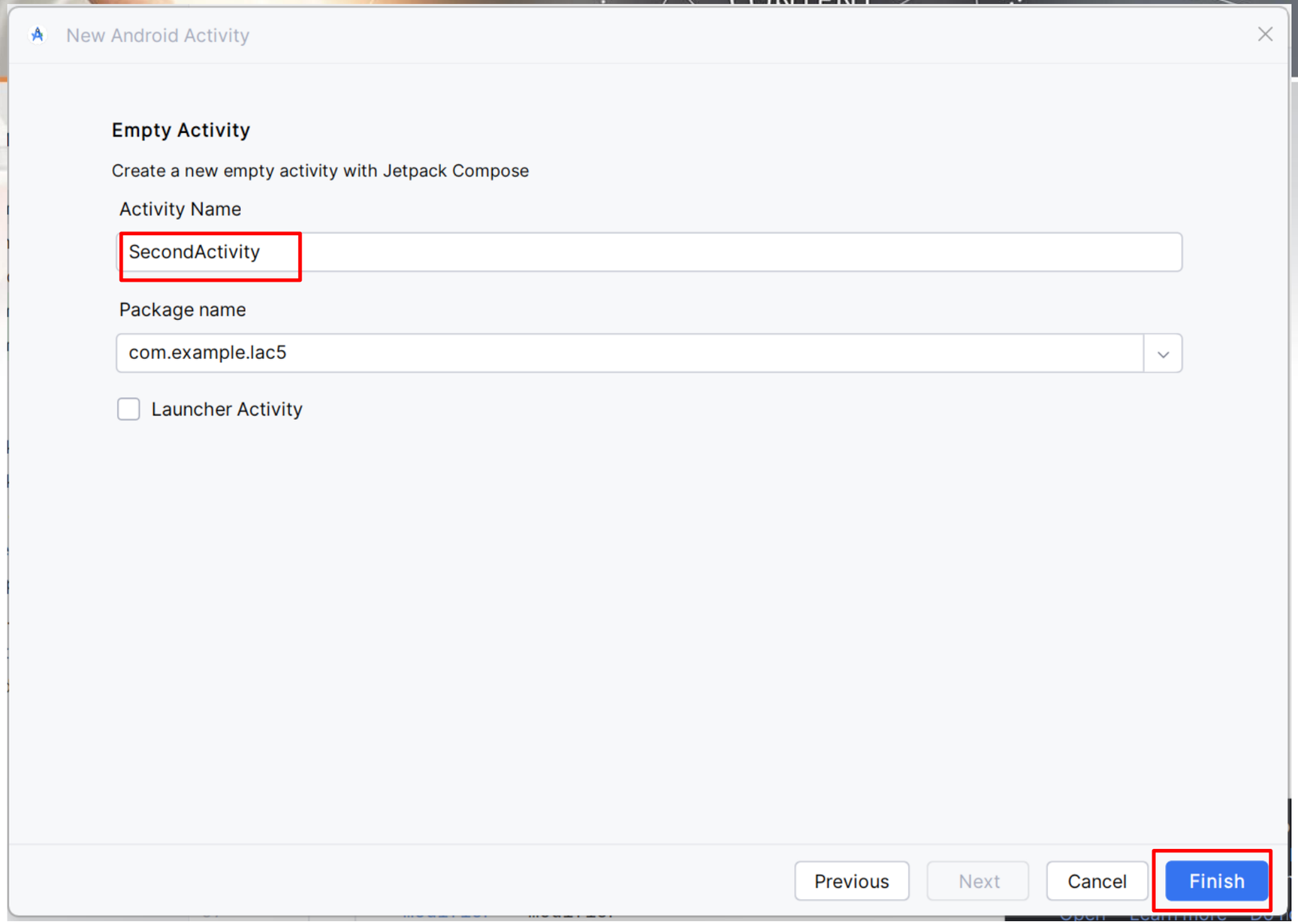
Intent Example

Add a new Activity: SecondActivity



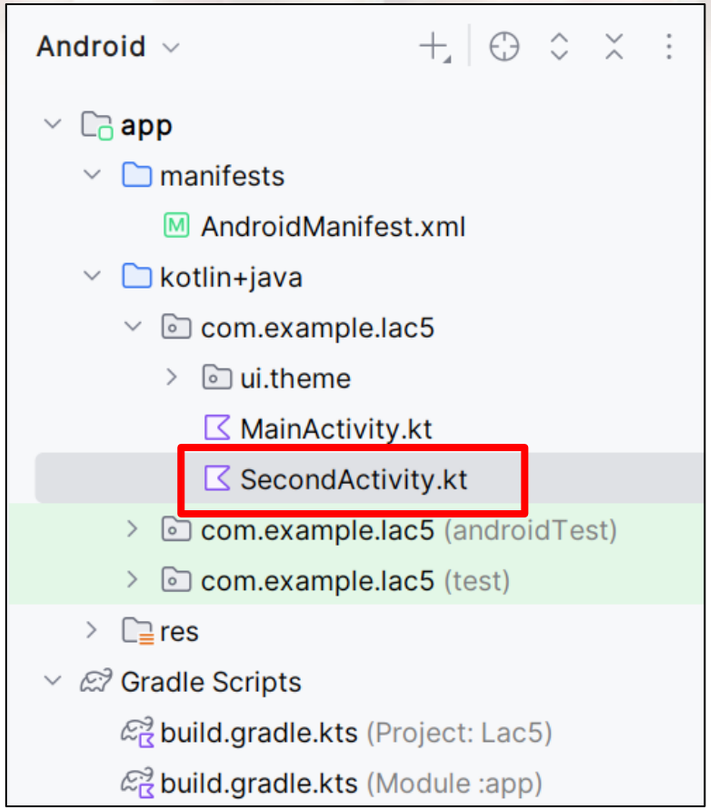
Intent Example

Add Activity



Add Activity (cont.)

app>> manifests >> AndroidManifest.xml




```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Lac5"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Lac5">
        <activity
            android:name=".SecondActivity"
            android:exported="false"
            android:label="SecondActivity"
            android:theme="@style/Theme.Lac5" />
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="Lac5"
            android:theme="@style/Theme.Lac5">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

Second Activity

For Launcher Activity

Intents Example

Intents Example (cont.)

MainActivity.kt

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Lac5Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    MyPage1(
                        modifier = Modifier.padding( paddingValues = innerPadding)
                    )
                }
            }
        }
    }
}
```

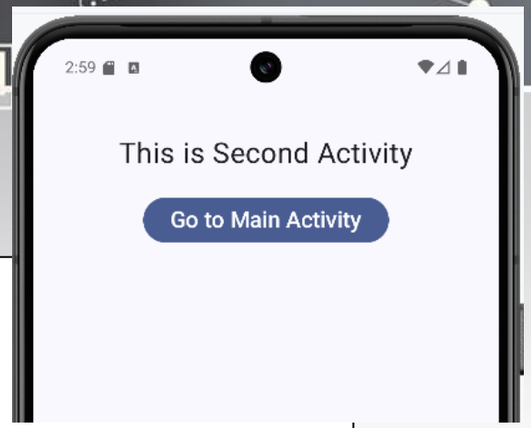
```
@Composable
fun MyPage1(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    Column(
        modifier = modifier
            .fillMaxSize()
            .verticalScroll( state = rememberScrollState())
            .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(
            text = "This is Page1",
            fontSize = 25.sp
        )
        Button(modifier = Modifier.padding( all = 20.dp),
            onClick = {
                context.startActivity( p0 = Intent( packageContext = context,
                    cls = SecondActivity::class.java))
            }
        ) {
            Text(text = "Go to Second Activity",
                fontSize = 20.sp)
        }
    }
}
```

Intents Example (cont.)

SecondActivity.kt

```
class SecondActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Lac5Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    MyPage2(
                        modifier = Modifier.padding( paddingValues = innerPadding)
                    )
                }
            }
        }
    }
}
```

```
@Composable
fun MyPage2(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    Column(
        modifier = modifier
            .fillMaxSize()
            .verticalScroll( state = rememberScrollState())
            .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(
            text = "This is Second Activity",
            fontSize = 25.sp        )
        Button(modifier = Modifier.padding( all = 20.dp),
            onClick = {
                context.startActivity( p0 = Intent( packageContext = context,
                    cls = MainActivity::class.java))
                ///  Close Activity
                (context as? Activity)?.finish()  }
        ) {
            Text(text = "Go to Main Activity",
                fontSize = 20.sp)
        }
    }
}
```



Intents: Open Another App

- Intents can start an activity in another app

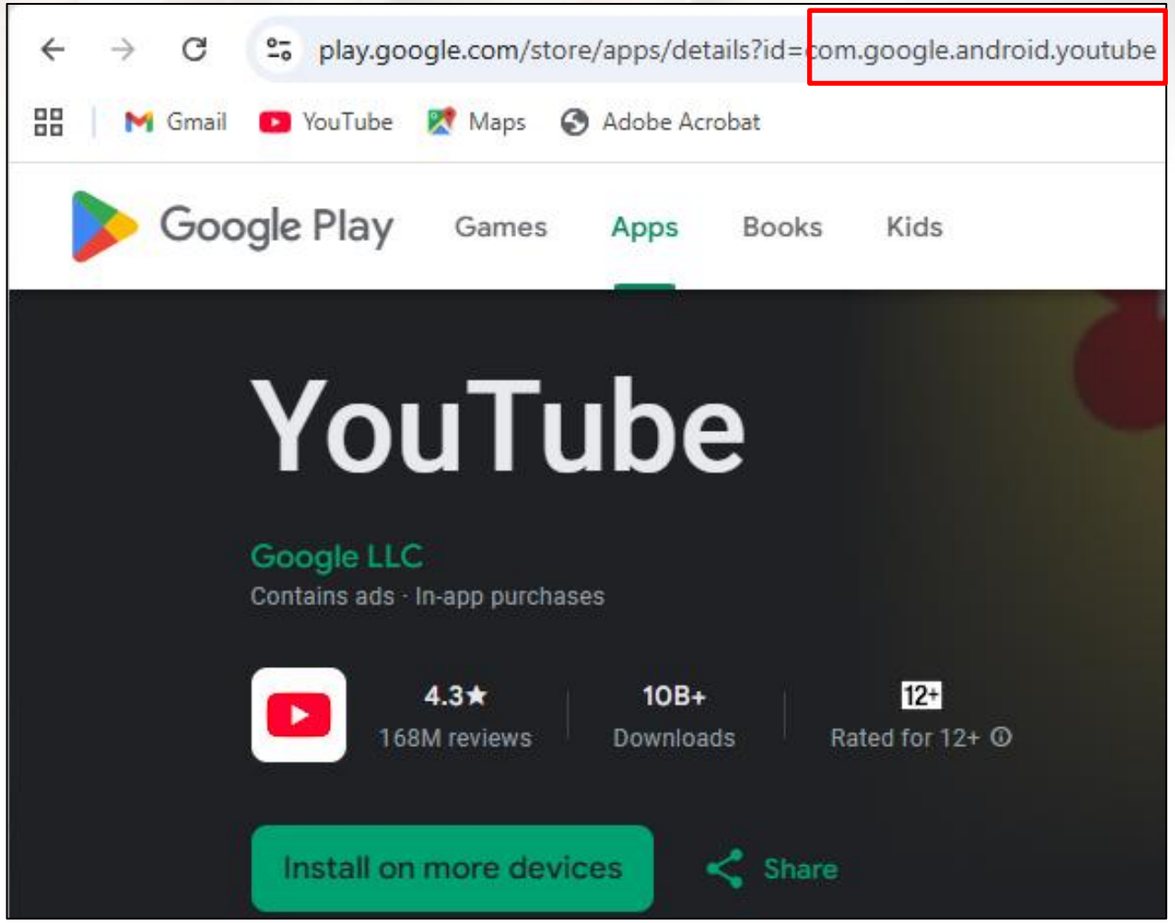
```
val i = Intent(Intent.ACTION_MAIN)
        i.package= packageName
        context?.startActivity(i)
```

- No app on Mobile phone : NullPointerException

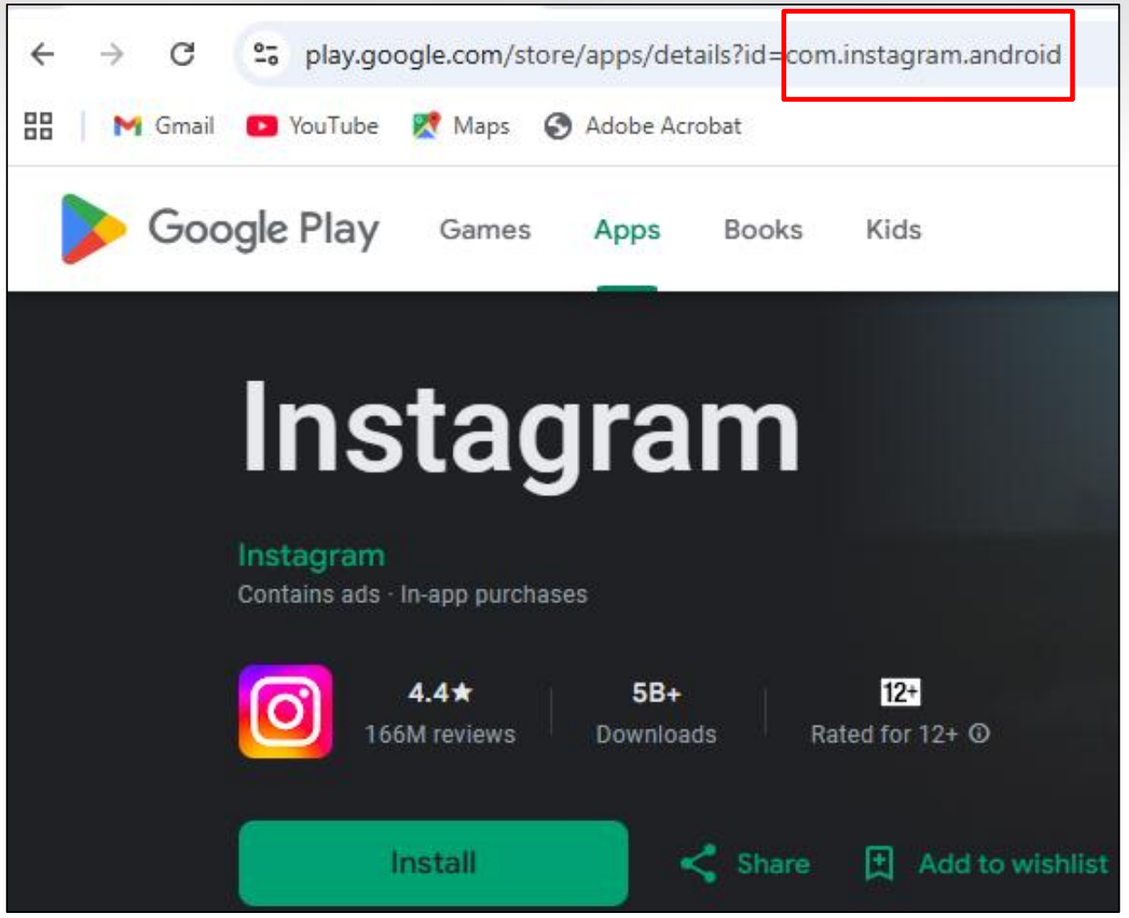
```
val i = Intent(Intent.ACTION_VIEW)
        i.data = Uri.parse("https://play.google.com/store/apps/details?id=$packageName")
        ContextCompat.startActivities(context!!,arrayOf(i), null)
```

Package Name

id=com.google.android.youtube

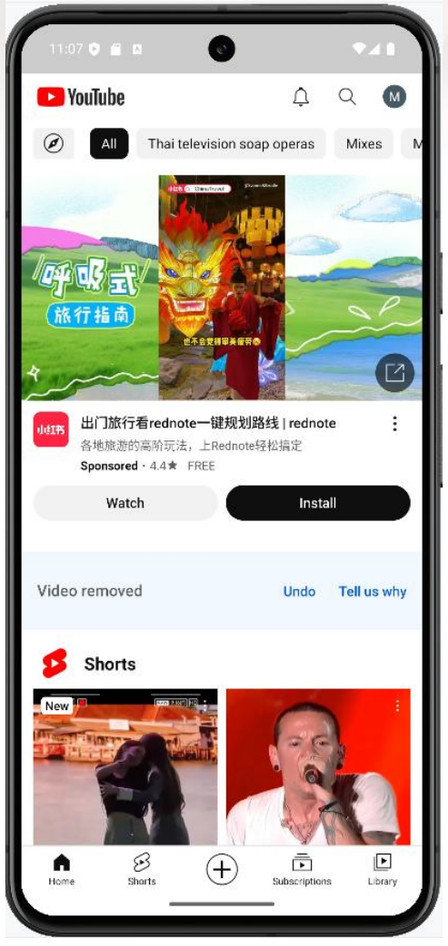


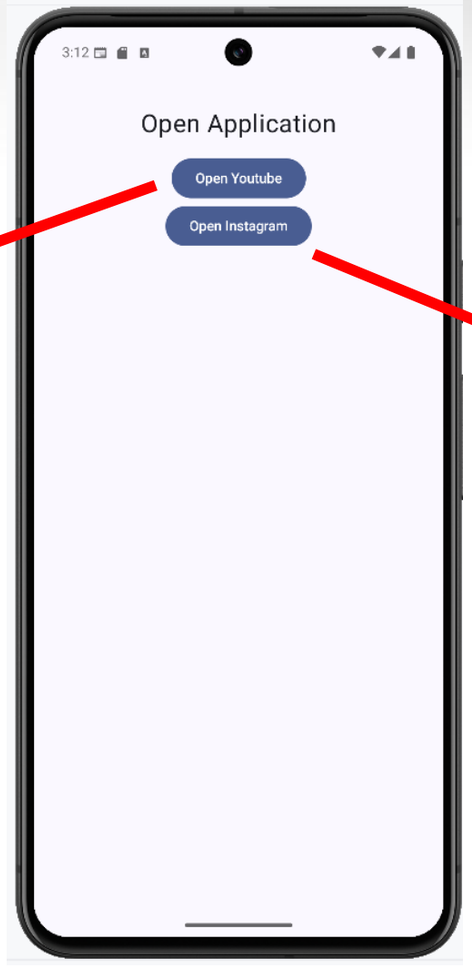
id=com.instagram.android

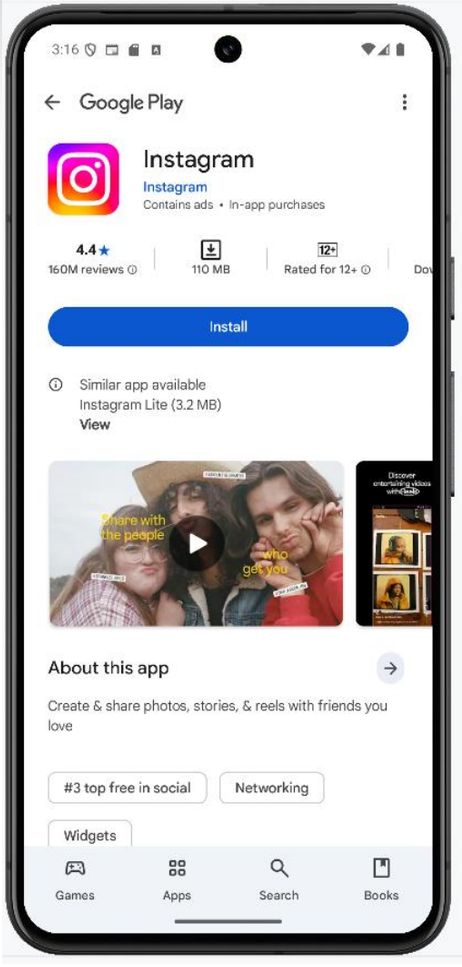


Intents: Open Another App

NullPointerException

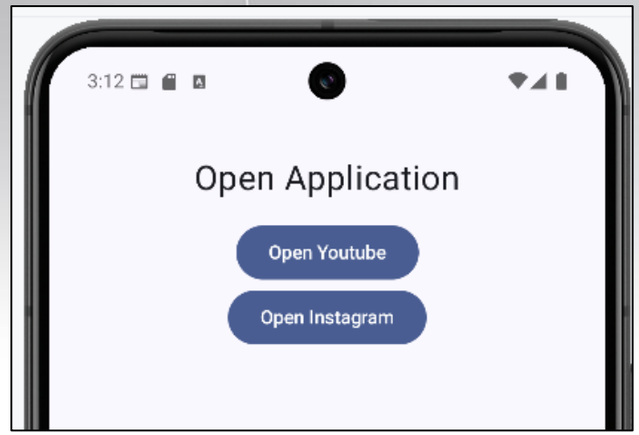






Intents: Open Another App

MainActivity.kt

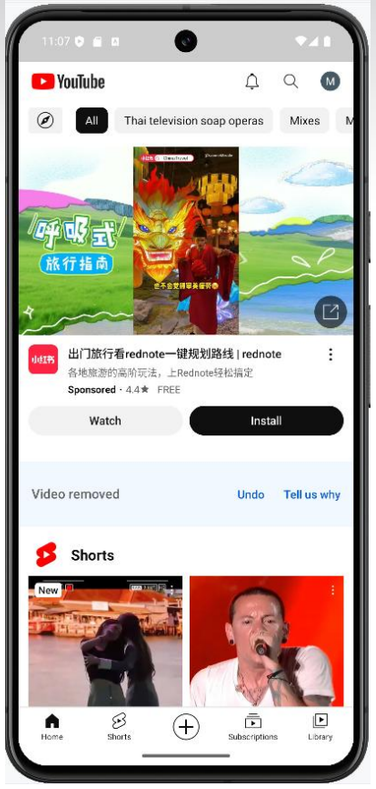


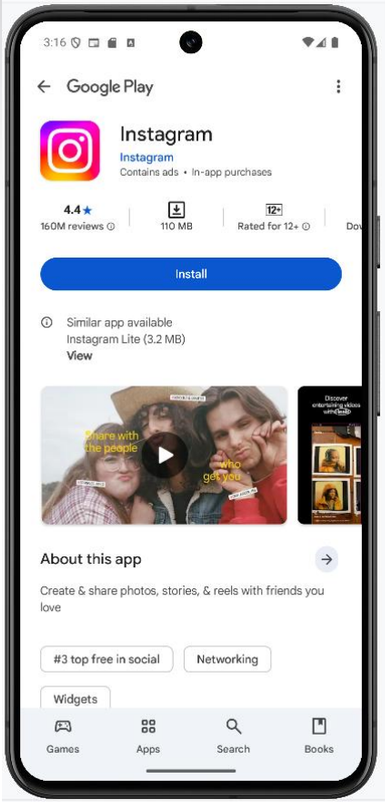

```
@Composable
fun AnotherApp(modifier: Modifier = Modifier){
    val context = LocalContext.current
    Column(modifier = modifier
        .fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Text( modifier = Modifier
            .padding( all = 16.dp),
            text = "Open Application",
            fontSize = 25.sp
        )
        //// Open Youtube Button
        Button(
            onClick = {
                val packageName = "com.google.android.youtube"
                startActivitySafe(context, packageName = packageName)
            }
        ) {
            Text(text = "Open Youtube")
        //// Open Instagram Button
        Button(
            onClick = {
                val packageName = "com.instagram.android"
                startActivitySafe(context, packageName = packageName)
            }
        ) {
            Text(text = "Open Instagram")
        }
    }
}
```

Intents: Open Another App

MainActivity.kt (cont)

```
fun startActivitySafe(context: Context?, packageName:String) {
    try {
        val intent = Intent( action = Intent.ACTION_MAIN)
        intent.`package`=packageName
        context?.startActivity( p0 = intent)
    } catch (e: Exception) {
        // Handle any exceptions
        val i = Intent( action = Intent.ACTION_VIEW)
        i.data =("https://play.google.com/store/apps/details?id=$packageName").toUri()
        ContextCompat.startActivities(context!!, intents = arrayOf(i) , options = null)
    }
}
```



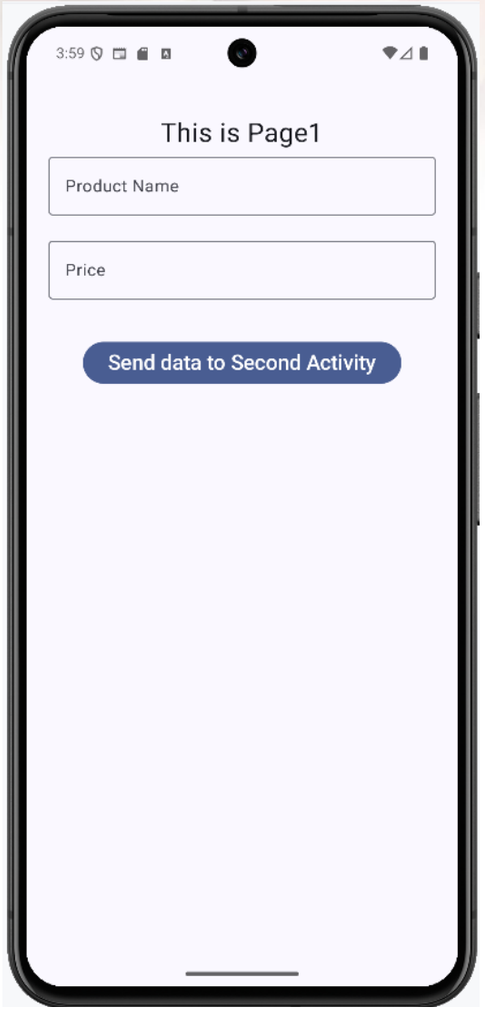


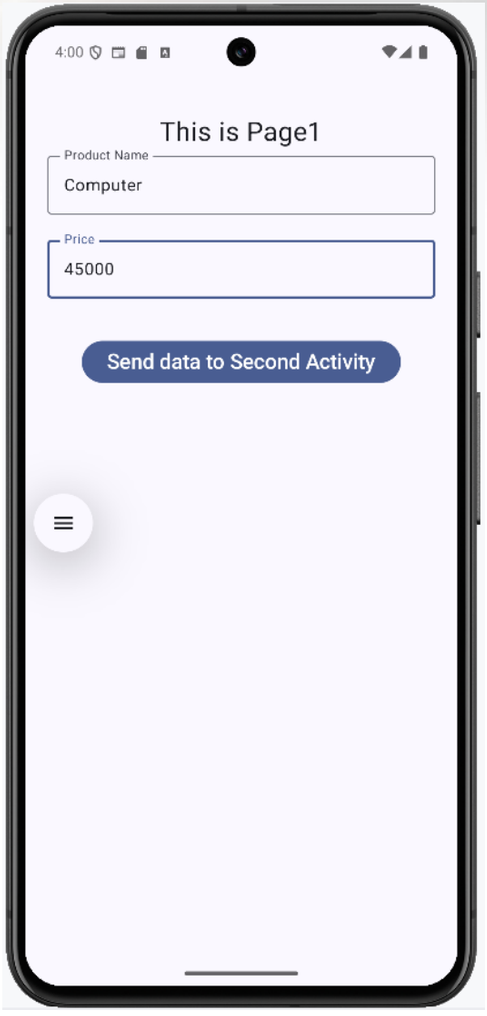
Intents with Parameter

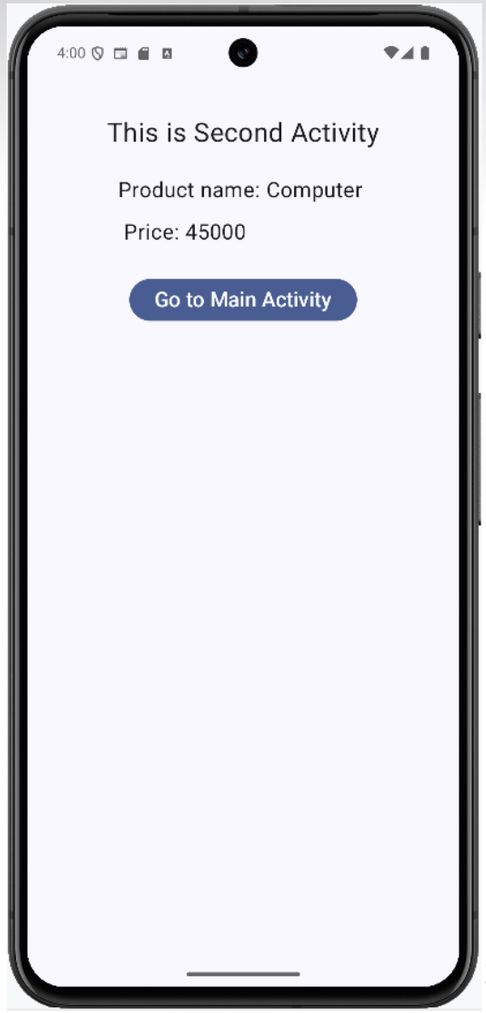
- To pass data onto the new activities, we use key value pairs inside the function putExtra, putStringArrayListExtra etc.
- putExtra generally passes the basic types such as Int, Float, Char, Double, Boolean, String along with

```
val i = Intent(this, OtherActivity::class.java)
i.putExtra("keyString", value)
context.startActivity(i)
```

Intents with Parameter



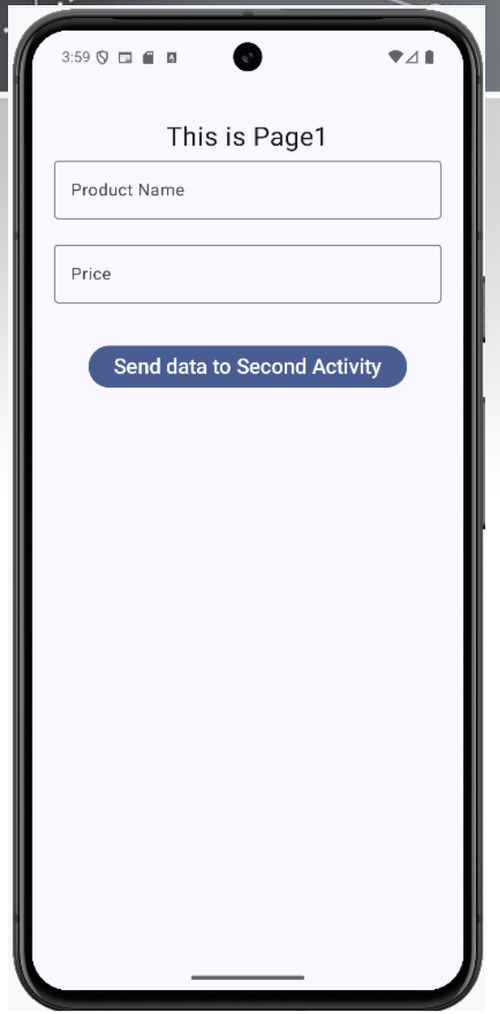




Intent with Parameter

MainActivity.kt

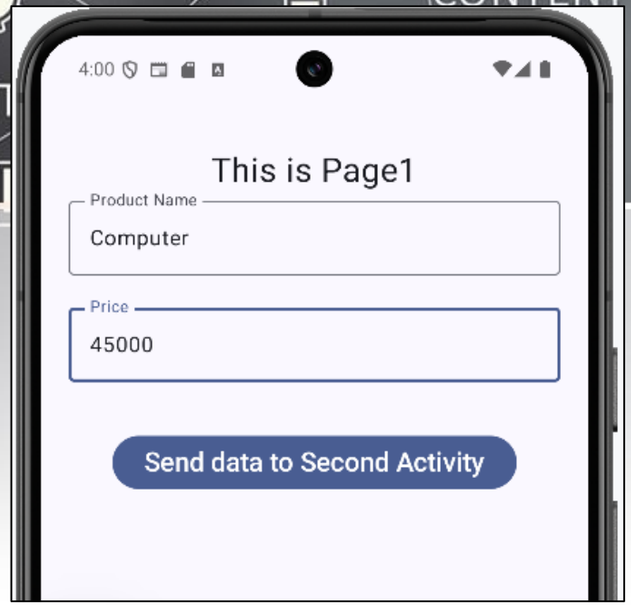
```
@Composable
fun ProductContent(name: String, onNameChange:(String) -> Unit,
                   price: String, onPriceChange:(String) -> Unit)
{
    Column(modifier = Modifier.padding(horizontal = 5.dp, )) {
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = name,
            onValueChange = onNameChange,
            label = { Text( text = "Product Name") } )
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = price,
            onValueChange = onPriceChange,
            label = { Text( text = "Price") })
    }
}
```



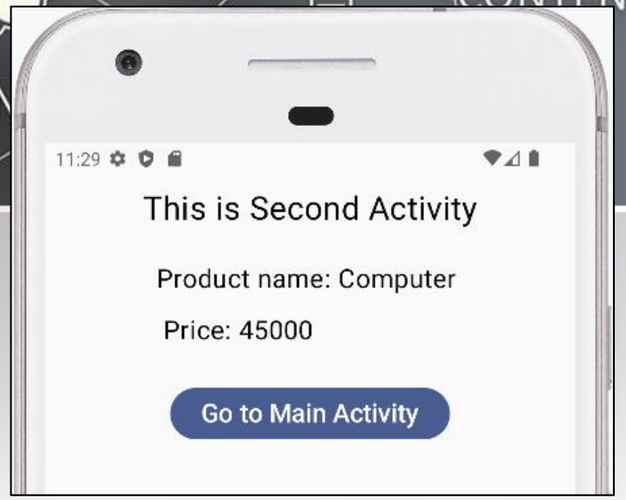
Intent with Parameter

MainActivity.kt(cont.)

```
@Composable
fun MyPage1(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    var name by rememberSaveable { mutableStateOf( value = "") }
    var price by rememberSaveable { mutableStateOf( value = "") }

    Column(
        modifier = modifier
            .fillMaxSize()
            .verticalScroll( state = rememberScrollState())
            .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(
            text = "This is Page1",
            fontSize = 25.sp
        )

        //Call ProductContent function
        ProductContent(name= name, onNameChange = { name= it },
            price = price, onPriceChange = { price = it })
        //Call ProductContent function
        ProductContent(name= name, onNameChange = { name= it },
            price = price, onPriceChange = { price = it })

        Button(modifier = Modifier.padding( all = 20.dp),
            onClick = {
                val i = Intent( packageContext = context, cls = SecondActivity::class.java)
                i.putExtra( name = "name", value = name)
                i.putExtra( name = "price", value = price.toInt())
                context.startActivity( p0 = i)
            }
        ) {
            Text(text = "Send data to Second Activity",
                fontSize = 20.sp)
        }
    }
}
```



Intent with Parameter



SecondActivity.kt

```
@Composable
fun MyPage2(modifier: Modifier = Modifier) {
    val context = LocalContext.current
    val activity = context.findActivity()
    val intent = activity?.intent
    val mName = intent?.getStringExtra( name = "name")?: "No Name"
    val mPrice = intent?.getIntExtra( name = "price", defaultValue = 0)

    Column( modifier = modifier
        .fillMaxSize()
        .verticalScroll( state = rememberScrollState())
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Text( modifier = Modifier.padding( all = 20.dp),
            text = "This is Second Activity",
            fontSize = 25.sp         )
        Text(text = "Product name: $mName \n Price: $mPrice",
            lineHeight = 40.sp   ,
            fontSize = 20.sp     )
        Button(modifier = Modifier.padding( all = 20.dp),
            onClick = {
                context.startActivity( p0 = Intent( packageContext = context,
                    cls = MainActivity::class.java))
                ///  Close Activity
                activity?.finish() }
        ) {
            Text(text = "Go to Main Activity",
                fontSize = 20.sp)
        }
    }
}
```

```
fun Context.findActivity(): Activity? = when (this) {
    is Activity -> this
    is ContextWrapper -> baseContext.findActivity()
    else -> null
}
```

Jetpack Navigation

- Navigation Component is library that helps to navigate between multiple screens.
- First, add a dependency on navigation in build.gradle file and Sync.

```
implementation(libs.androidx.navigation.compose)
```

Jetpack Navigation

- 3 main parts of Navigation component:
  1. Navigation graph
  2. NavHostController
  3. NavHost

Jetpack Navigation

1. Navigation graph

– It that contains all navigation-related information in one centralized location.

– This includes all of the individual content areas within your app, called destinations, as well as the possible paths that a user can take through your app.

Jetpack Navigation

2. NavHostController: the central API for the Navigation component. It is stateful and keeps track of the back stack of composables that make up the screens of your app.

Key Features:

- Navigation Management: Provides methods to navigate to destinations (e.g., .navigate()) or handle back actions (e.g., .popBackStack()).
- Back Stack Management: Automatically maintains the history of visited screens, ensuring the 'Back' button works correctly.
- State & Data Holder: Acts as a store for screen states (ViewModelStoreOwner) and facilitates passing arguments/data between destinations via SavedStateHandle.

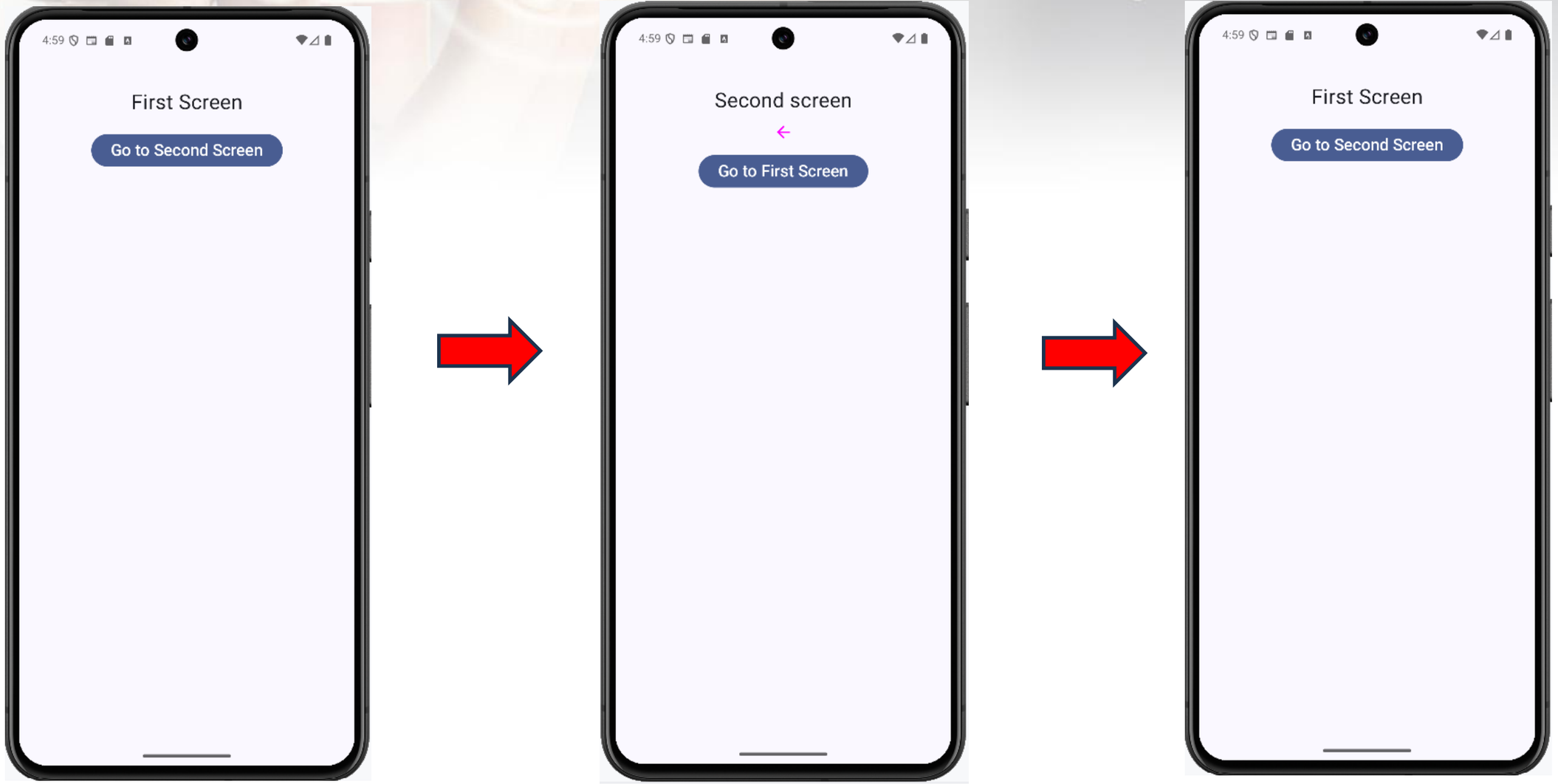
Jetpack Navigation

3. NavHost: a composable container that displays the current screen based on the Navigation Graph. It swaps destinations in and out as the user navigates.

NavHost takes three main parameters inside it.

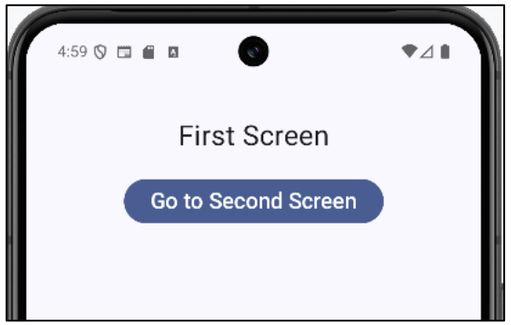
- navHostController : The controller that manages app navigation and the back stack.
- startDestination : The route (String) of the first screen to display when the app launches.
- NavGraphBuilder.() : A lambda where you define the navigation graph and all composable destinations.

Navigating Between Screens



Navigating Between Screens

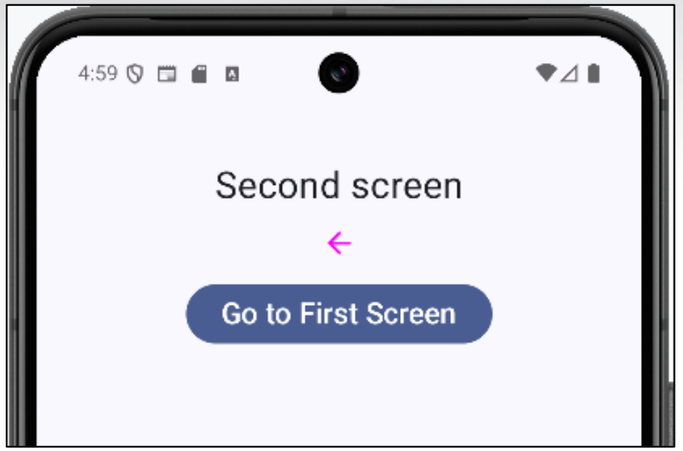
```
@Composable
fun ComposeNavigation(modifier : Modifier = Modifier) {
    val navController = rememberNavController()
    Column(
        modifier = modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        NavHost(navController = navController, startDestination = "First"){
            composable( route = "First"){
                FirstScreen( navHostController = navController)
            }
            composable( route = "Second"){
                SecondScreen( navHostController = navController)
            }
        }
    }
}
```




```
@Composable
fun FirstScreen(navHostController: NavHostController) {
    Column(modifier = Modifier
        .fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally)
    {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(text = "First Screen",
            fontSize = 25.sp)
        Spacer(modifier = Modifier.height(height = 20.dp))
        Button(onClick = {
            navHostController.navigate( route = "Second")
        }) {
            Text(text = "Go to Second Screen",
                fontSize = 20.sp)
        }
    }
}
```

Navigating Between Screens

```
@Composable
fun SecondScreen(navHostController: NavHostController) {
    Column(modifier = Modifier
        .fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally) {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(text = "Second screen", fontSize = 25.sp)
        IconButton(onClick = {
            navHostController.navigateUp()
        }) {
            Icon(
                imageVector = Icons.AutoMirrored.Filled.ArrowBack,
                contentDescription = "", tint = Color.Magenta)
        }
        Button(onClick = {
            navHostController.navigate( route = "First") {
                popUpTo( route = "First") { inclusive = true } //Clear Stack
                launchSingleTop = true}
        }) {
            Text(text = "Go to First Screen", fontSize = 20.sp)
        }
    }
}
```

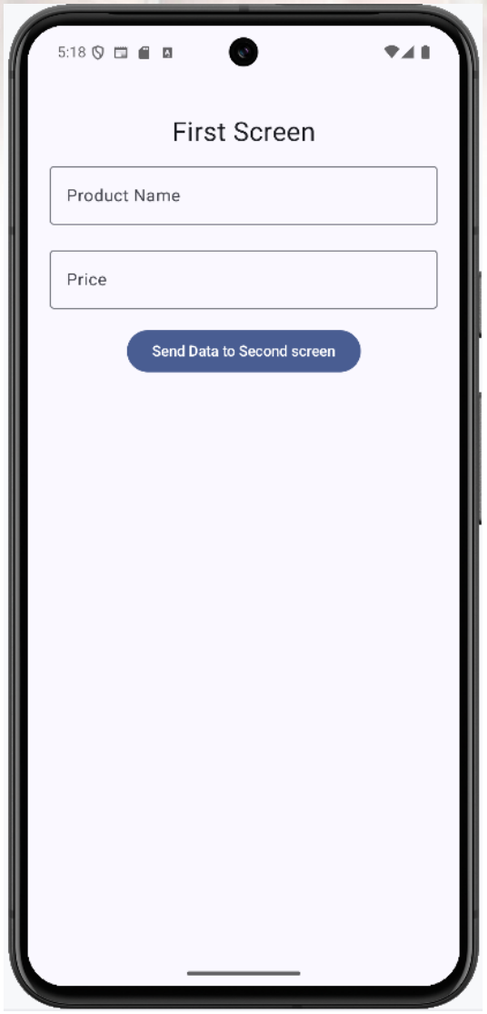


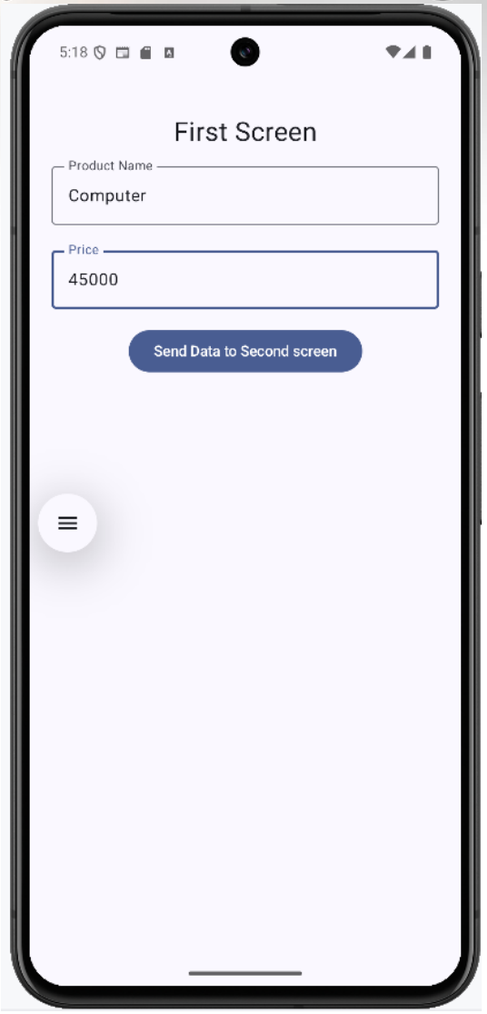
Passing data between screen

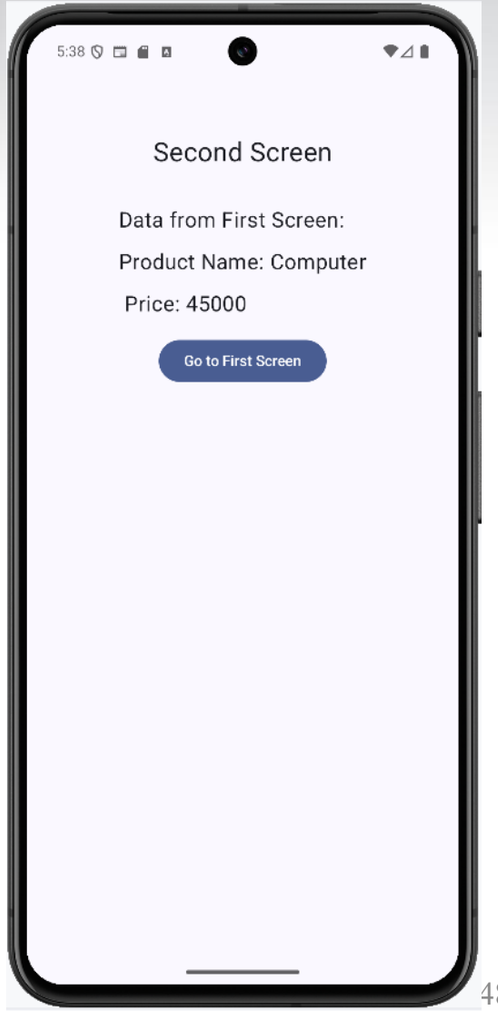
Methods for passing data between screen:

1. Route Arguments (Basic Data)
    - Pass simple data directly in the route path.
    - Example: "detail_screen/{itemId}“
    - Use case: Passing IDs, Names.
2. SavedStateHandle (Complex Data)
    - Pass Objects (Parcelable) via NavHostController.
    - Uses savedStateHandle to store data between screens.
    - Use case: Passing full User object, Search results.

Passing data by Route Arguments



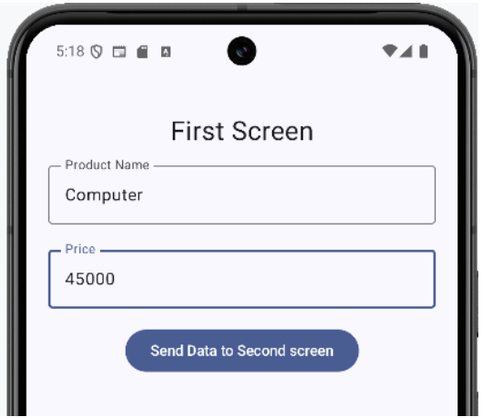


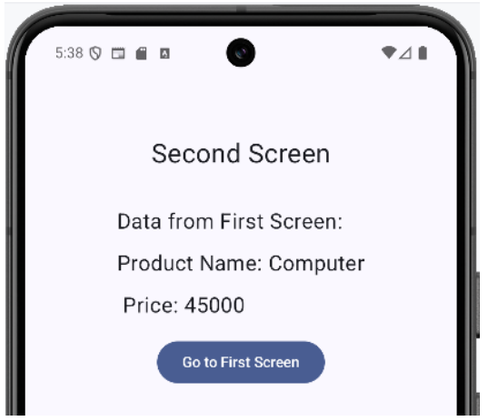


Passing data by Route Arguments

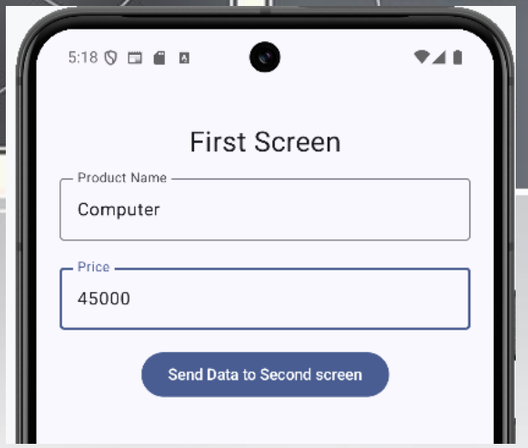
```
sealed class Screen(val route: String) {
    4 Usages
    data object First : Screen( route = "first_screen")
    2 Usages
    data object Second : Screen( route = "second_screen"+"/{name}"+"/{price}") {
        1 Usage
        fun sendName(name: String, price: String) = "second_screen/$name/$price"
    }
}
```

```
@Composable
fun ComposeNavigation(modifier : Modifier = Modifier) {
    val navController = rememberNavController()
    Column(
       modifier = modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {  NavHost(navController = navController,
        startDestination = Screen.First.route) {
            composable(Screen.First.route) {
                FirstScreen( navHostController = navController)
            }
            composable(Screen.Second.route) {
                val name = it.arguments?.getString( key = "name") ?: "no name"
                val price = it.arguments?.getString( key = "price") ?: "0"
                SecondScreen( navHostController = navController, name, price)
            }
        }
    }
}
```





Passing data by Route Arguments

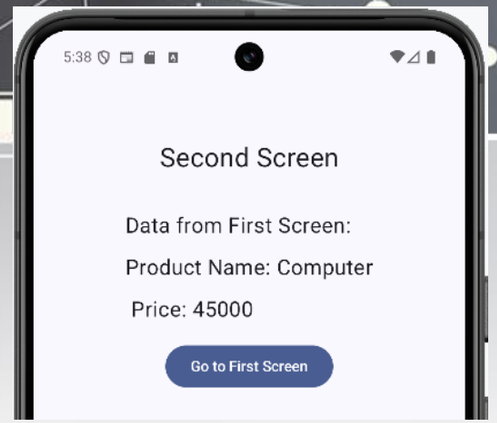


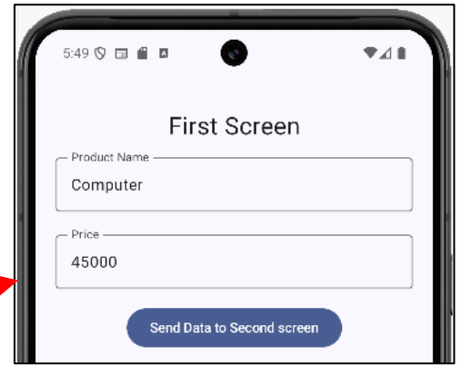

```
@Composable
fun FirstScreen(navHostController: NavHostController) {
    var name by rememberSaveable { mutableStateOf( value = "") }
    var price by rememberSaveable { mutableStateOf( value = "") }
    Column(modifier = Modifier.fillMaxSize().padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally)
    {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(text = "First Screen", fontSize = 25.sp)
        Spacer(modifier = Modifier.height(height = 10.dp))
        //Call ProductContent function
        ProductContent(name= name, onNameChange = { name= it },
            price = price, onPriceChange = { price = it })
        Button(onClick = {
            navHostController.navigate( route = Screen.Second.sendName(name, price))
        }) {
            Text(text = "Send Data to Second screen")
        }
    }
}
```

```
@Composable
fun ProductContent(name: String, onNameChange:(String) -> Unit,
                   price: String, onPriceChange:(String) -> Unit)
{
    Column(modifier = Modifier.padding(horizontal = 5.dp, )) {
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = name,
            onValueChange = onNameChange,
            label = { Text( text = "Product Name") } )
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = price,
            onValueChange = onPriceChange,
            label = { Text( text = "Price") })
    }
}
```

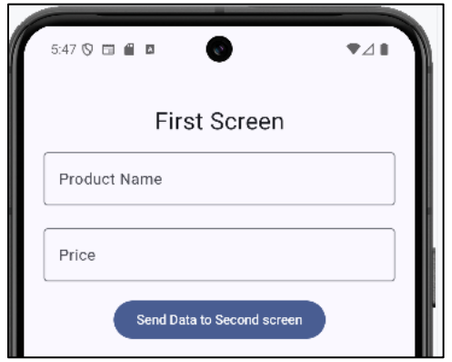
Passing data by Route Arguments

```
@Composable
fun SecondScreen(
    navHostController: NavHostController,
    name: String, price: String) {
    Column( modifier = Modifier.fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally)
    {
        Spacer(modifier = Modifier.height(height = 20.dp))
        Text(modifier = Modifier.padding( all = 20.dp),
            text = "Second Screen", fontSize = 25.sp)
        Spacer(modifier = Modifier.height(height = 10.dp))
        Text(text = "Data from First Screen:\nProduct Name: $name\n Price: $price",
            lineHeight = 40.sp  ,fontSize = 20.sp)
        Spacer(modifier = Modifier.height(height = 10.dp))
        Button(onClick = {
            // navHostController.navigateUp()
            navHostController.navigate(Screen.First.route) {
                popUpTo(Screen.First.route) { inclusive = true } //Clear Stack
                launchSingleTop = true}
        }) {
            Text(text = "Go to First Screen")
        }
    }
}
```





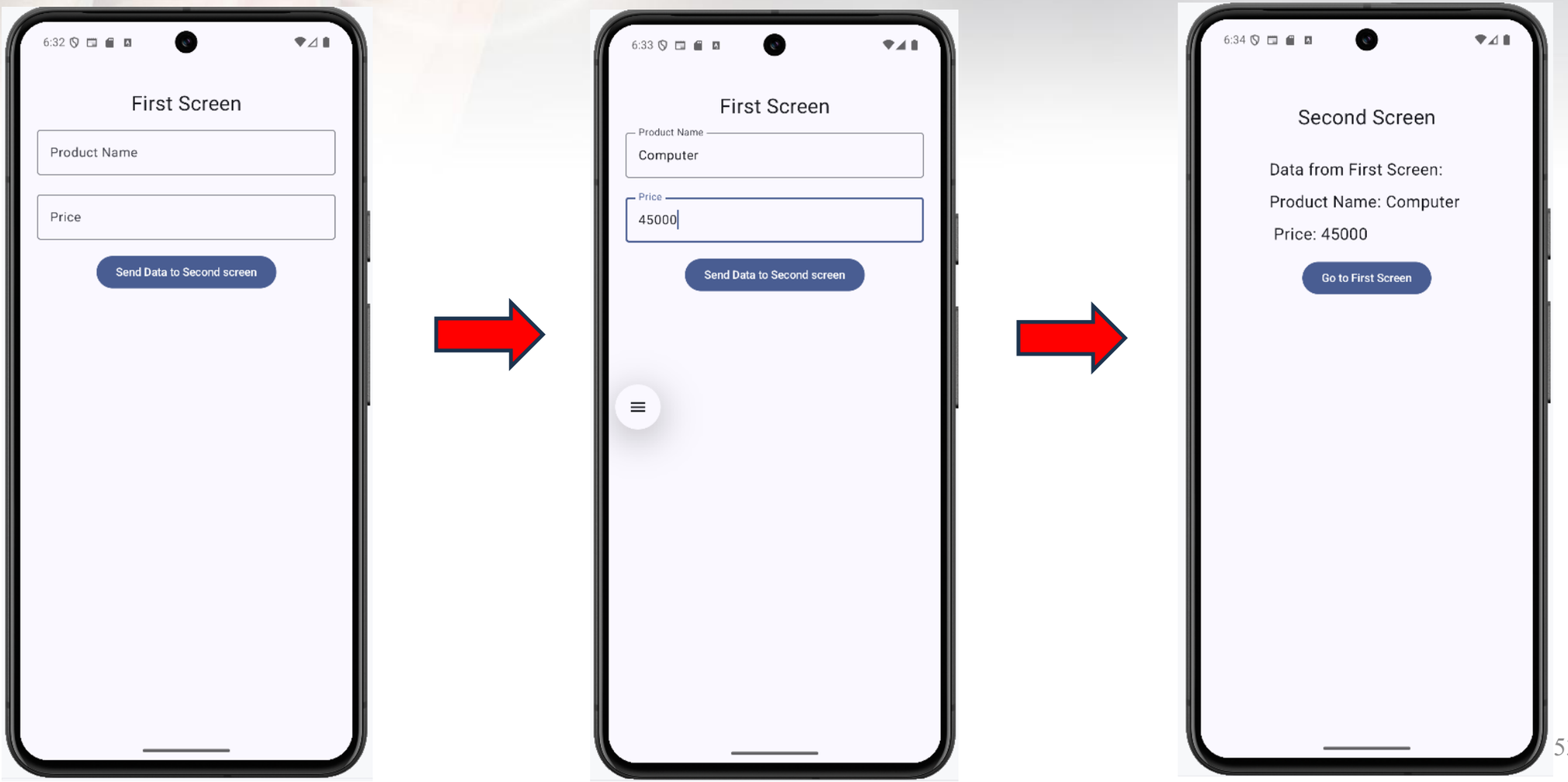
navHostController.popBackStack()



Passing data by Through SavedStateHandle

- Parcelable is interface for passing data as an object between android application components.
- Parcelize support: The kotlin-parcelize plugin provides an automatic Parcelable implementation generator using the @Parcelize annotation.

Passing data by Through SavedStateHandle



Passing data by through SavedStateHandle

- First, add kotlin-parcelize plugin in build.gradle file and Sync.

id ("kotlin-parcelize")

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)

    id ("kotlin-parcelize")
}
```

Passing data by through SavedStateHandle

- Create the data class and parcelize it.

```
@Parcelize
data class Product(
    val name: String,
    val price: Int
): Parcelable
```

- Define the routes through sealed class and pass it in the NavHost composable function.

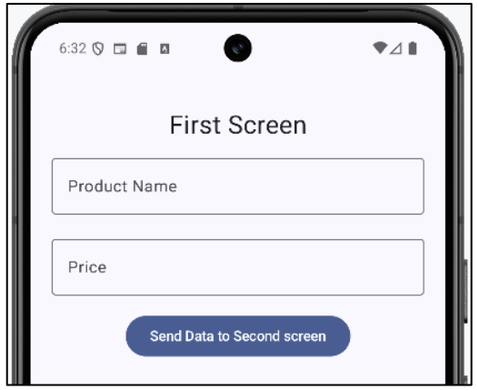
```
sealed class Screen(val route: String) {
    2 Usages
    data object First : Screen( route = "first_screen")
    2 Usages
    data object Second : Screen( route = "second_screen")
}
```

Passing data by through SavedStateHandle

```
@Composable
fun ComposeNavigation(modifier : Modifier = Modifier) {
    val navController = rememberNavController()
    Column( modifier = modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally ) {
        NavHost(
            navController = navController,
            startDestination = Screen.First.route
        ) {
            composable(Screen.First.route) {
                FirstScreen( navHostController = navController)
            }
            composable(Screen.Second.route) {
                SecondScreen( navHostController = navController)
            }
        }
    }
}
```

Passing data between screen by through SavedStateHandle

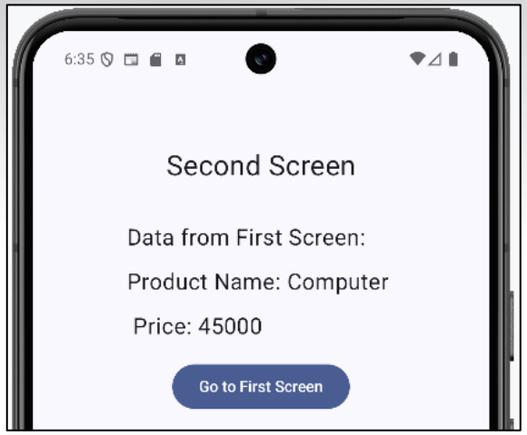
```
@Composable
fun FirstScreen( navHostController: NavHostController) {
    var name by rememberSaveable { mutableStateOf( value = "") }
    var price by rememberSaveable { mutableStateOf( value = "") }
    Column(modifier = Modifier
        .fillMaxSize()
        .padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally)
    {   Spacer(modifier = Modifier.height(height = 20.dp))
        Text(text = "First Screen", fontSize = 25.sp)
        Spacer(modifier = Modifier.height(height = 10.dp))
```

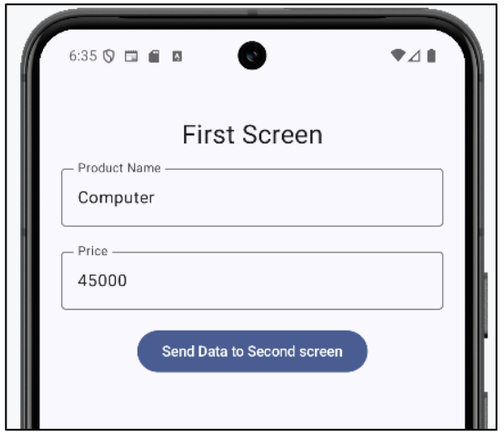



```
        //Call ProductContent function
         ProductContent(name= name, onNameChange = { name= it },
            price = price, onPriceChange = { price = it })
        Button(onClick = {
            navHostController.currentBackStackEntry?.savedStateHandle?.set(
                "data",
                Product(name, price.toInt()))
            navHostController.navigate(Screen.Second.route)
        }) {
            Text(text = "Send Data to Second screen")
        }
    }
}
```

Passing data between screen by through SavedStateHandle

```
@Composable
fun SecondScreen(navHostController: NavHostController) {
    val data = navHostController.previousBackStackEntry?.savedStateHandle?.get<Product>("data")
        ?: Product(name = "", price = 0)
    Column(
        modifier = Modifier.fillMaxSize().padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height( height = 20.dp))
        Text( modifier = Modifier.padding( all = 20.dp),
            text = "Second Screen",fontSize = 25.sp )
        Spacer(modifier = Modifier.height( height = 10.dp))
        Text( text = "Data from First Screen:\nProduct Name: ${data.name} " +
                "\n Price: ${data.price}",
            lineHeight = 40.sp, fontSize = 20.sp )
        Spacer(modifier = Modifier.height(height = 10.dp))
        Button(onClick = {
           navHostController.navigateUp()
        }) {
            Text(text = "Go to First Screen")
        }
    }
}
```

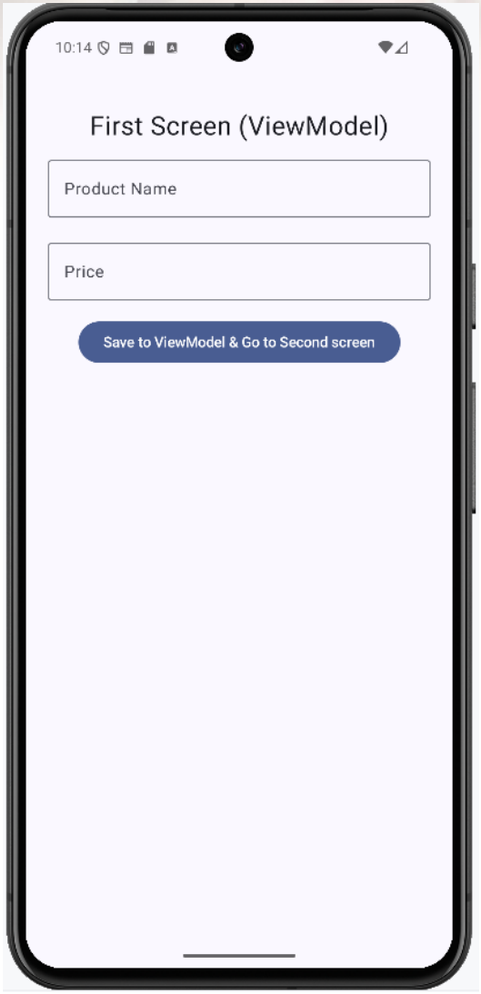


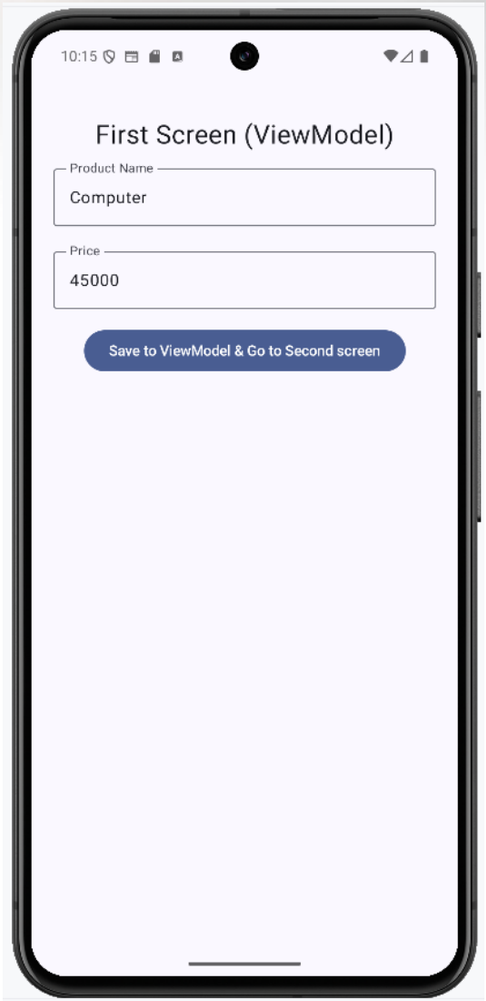


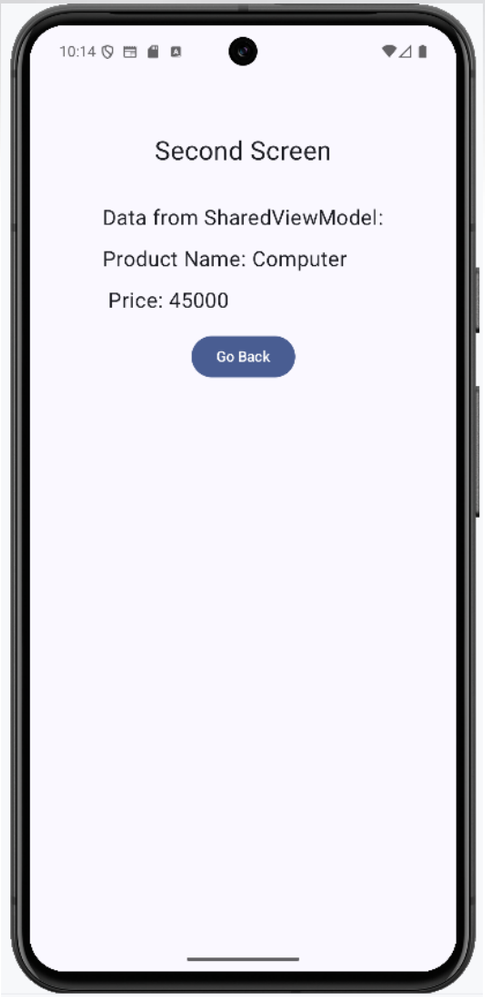
Shared ViewModel (Centralized Data)

- Instead of passing data through a chain of screens (A -> B -> C), all screens observe data from a single shared ViewModel.
- Key Advantages of Shared ViewModel:
  - Streamlines Data Access: Direct access to data without passing through intermediate screens.
  - Live Updates: If Screen B updates the data, Screen A (in the back stack) gets the updated data automatically immediately.
  - Complex Data Handling: It handles large objects or lists much better than Parcelable in arguments, which has size limits.
  - Data Persistence: The data remains safe in the ViewModel even if the screen rotates or the configuration changes.

Shared ViewModel (Centralized Data)







Shared ViewModel (Centralized Data)

- Create the Shared ViewModel

```
class SharedViewModel : ViewModel() {
    var product by mutableStateOf( value = Product( name = "", price = 0))
        private set

    fun updateProduct(name: String, price: Int) {
        product = Product(name, price)
    }
}
```

Shared ViewModel (Centralized Data)

- Navigation Graph

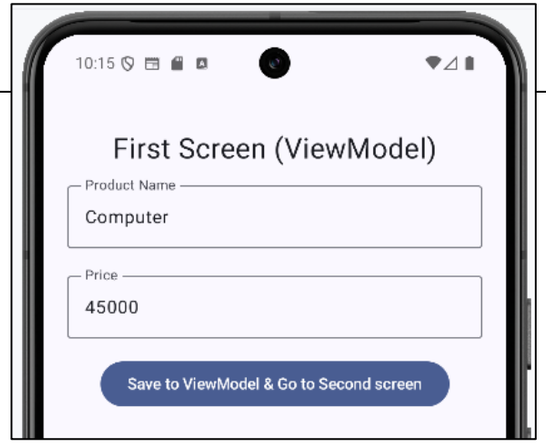
```
@Composable
fun ComposeNavigation(modifier : Modifier = Modifier) {
    val navController = rememberNavController()
    // Initialize the ViewModel here (Parent Scope) to share it across screens
    val sharedViewModel: SharedViewModel = viewModel()
    Column( modifier = modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally ) {
        NavHost(
            navController = navController,
            startDestination = Screen.First.route
        ) {
            composable(Screen.First.route) {
                // Pass the shared ViewModel instance to the FirstScreen
                FirstScreen( navHostController = navController, sharedViewModel)
            }
            composable(Screen.Second.route) {
                // Pass the same ViewModel instance to the SecondScreen
                SecondScreen( navHostController = navController, sharedViewModel)
            }
        }
    }
}
```

Shared ViewModel (Centralized Data)

- First Screen

```
@Composable
fun FirstScreen(
    navHostController: NavHostController,
    sharedViewModel: SharedViewModel // Get Shared ViewModel
) {
    // Local state for holding TextField inputs
    var name by rememberSaveable { mutableStateOf( value = "") }
    var price by rememberSaveable { mutableStateOf( value = "") }

    Column(modifier = Modifier.fillMaxSize().padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally)
    {   Spacer(modifier = Modifier.height( height = 20.dp))
        Text(text = "First Screen (ViewModel)", fontSize = 25.sp)
        Spacer(modifier = Modifier.height( height = 10.dp))
        ProductContent(name= name, onNameChange = { name= it },
            price = price, onPriceChange = { price = it })
```

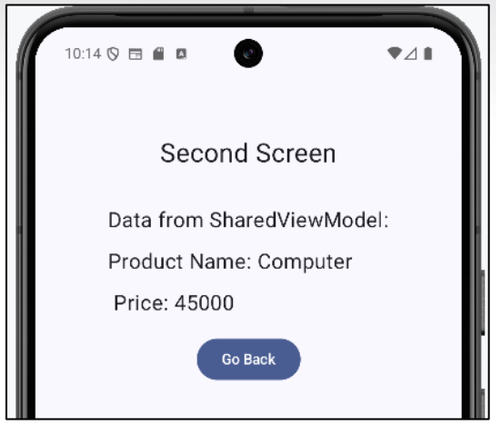
```
        Button(onClick = {
            // Save data to the Shared ViewModel
             val priceInt = price.toIntOrNull() ?: 0
            sharedViewModel.updateProduct(name, priceInt)
            // Navigate to the next screen (No arguments needed in the route)
            navHostController.navigate(Screen.Second.route)
        }) {
            Text(text = "Save to ViewModel & Go to Second screen")
        }
    }
}
```



Shared ViewModel (Centralized Data)

- Second Screen

```
@Composable
fun SecondScreen(
    navHostController: NavHostController,
    sharedViewModel: SharedViewModel // Get the Shared ViewModel
) {
    // Observe/Read data directly from the Shared ViewModel
    val data = sharedViewModel.product
    Column(
        modifier = Modifier.fillMaxSize().padding( all = 16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height( height = 20.dp))
        Text( modifier = Modifier.padding( all = 20.dp),
            text = "Second Screen",fontSize = 25.sp )
        Spacer(modifier = Modifier.height( height = 10.dp))
        // Display the data
        Text( text = "Data from SharedViewModel:\nProduct Name: ${data.name} " +
                "\n Price: ${data.price}",
            lineHeight = 40.sp, fontSize = 20.sp )
        Spacer(modifier = Modifier.height( height = 10.dp))
```




```
        Button(onClick = {
            navHostController.navigateUp()
        }) {
            Text(text = "Go Back")
        }
    }
}
```

Shared ViewModel (Centralized Data)

- The Product data class, Screen sealed class, and ProductContent function remain unchanged.

```
@Parcelize
data class Product(
    val name: String,
    val price: Int
): Parcelable

6 Usages  2 Inheritors
sealed class Screen(val route: String) {
    2 Usages
    data object First : Screen( route = "first_screen")
    2 Usages
    data object Second : Screen( route = "second_screen")
}
```

```
@Composable
fun ProductContent(name: String, onNameChange:(String) -> Unit,
                   price: String, onPriceChange:(String) -> Unit)
{
    Column(modifier = Modifier.padding(horizontal = 5.dp, )) {
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = name,
            onValueChange = onNameChange,
            label = { Text( text = "Product Name") } )
        OutlinedTextField(
            modifier = Modifier.width( width = 400.dp)
                .padding(bottom = 16.dp),
            value = price,
            onValueChange = onPriceChange,
            label = { Text( text = "Price") })
    }
}
```

References

- https://nameisjayant.medium.com/navigating-with-jetpack-compose-8ee83c32a1b9
- https://nameisjayant.medium.com/navigating-with-jetpack-compose-8ee83c32a1b9
- https://medium.com/@bhoomigadhiya/navigation-in-jetpack-compose-a-simple-guide-14f7caddbe1b
- https://medium.com/@bhoomigadhiya/navigation-in-jetpack-compose-a-simple-guide-14f7caddbe1b
- https://developer.android.com/jetpack/compose/navigation