IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT
2024 - Tutorial

ViewModel and Intents

ViewModel

ViewModel is a part of Android's Architecture Components, and it's designed to store and manage UI-related data. It survives configuration changes such as screen rotations, ensuring that data is preserved.

Why Use ViewModel?

Data Management: Retains data during configuration changes.

Lifecycle Awareness: ViewModel is designed to be lifecycle-aware, meaning it won't cause memory leaks and it'll be cleared once its associated lifecycle is destroyed.

Android Intents

In Android development, an Intent is a fundamental concept that facilitates communication between components of an application or between different applications. It represents an "intention" to perform an action.

Types of Intents:

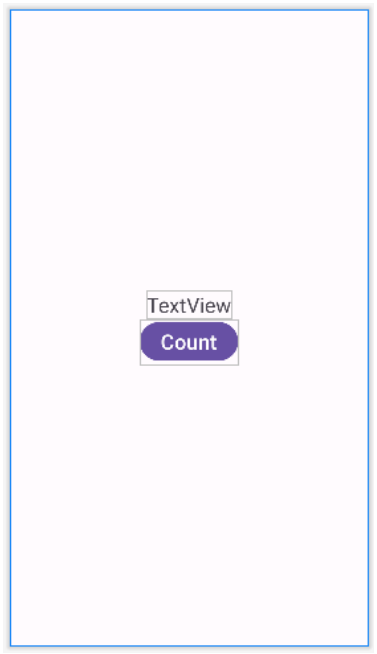
1. **Explicit Intents:** Specify the component to start with by name (i.e., you know the target). For example, you can use an explicit intent to start a service or to launch a particular activity within your application.
2. **Implicit Intents:** Do not name a specific component but instead declare a general action to perform, allowing a component from another app to handle it. For instance, if you want to show a webpage, you can use an implicit intent, and Android will determine the best app (like a browser) to display that webpage.

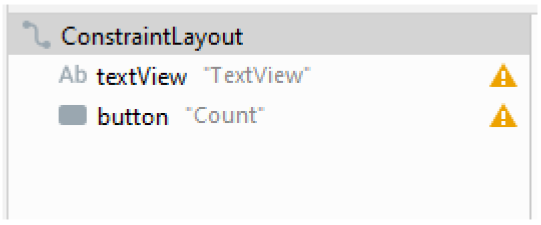
Key Features:

1. Navigation: You can use intents to navigate between activities within an app.
2. Data Transfer: Intents allow you to transfer data between components using extras. The data can be primitive data types, strings, or more complex objects.
3. Interacting with Other Apps: Using intents, your app can leverage other apps' capabilities. For example, you can use an intent to open the camera, and after taking a picture, the image data can be returned to your app.
4. Broadcasting Information: Intents can broadcast system-wide announcements (like battery low) to apps that might be interested in such information.

ViewModel Implementation

1. We will use the following UI to demonstrate it.





2. Implement the main activity as follows:

```
class MainActivity : AppCompatActivity() {
  var count = 0
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val textView:TextView = findViewById(R.id.textView)
    val button:Button = findViewById(R.id.button)
    textView.setText(count.toString())

    button.setOnClickListener {
      count++   number of times the button is pressed
      textView.setText(count.toString())

    }
  }
}
```

3. Now check the output.
4. See what happens when you rotate the screen.

5. Now let's implement the view model.
6. Create a new Kotlin class named MainActivityData and implement the following.

```
class MainActivityData:ViewModel() {

   private val _count = MutableLiveData<Int>().apply { value = 0 }

   val count:LiveData<Int> = _count       you can't change the value outside the class

   fun increment(){
      _count.value = _count.value!! + 1   use assert symbol- it is not going to be a null value
   }
}
```

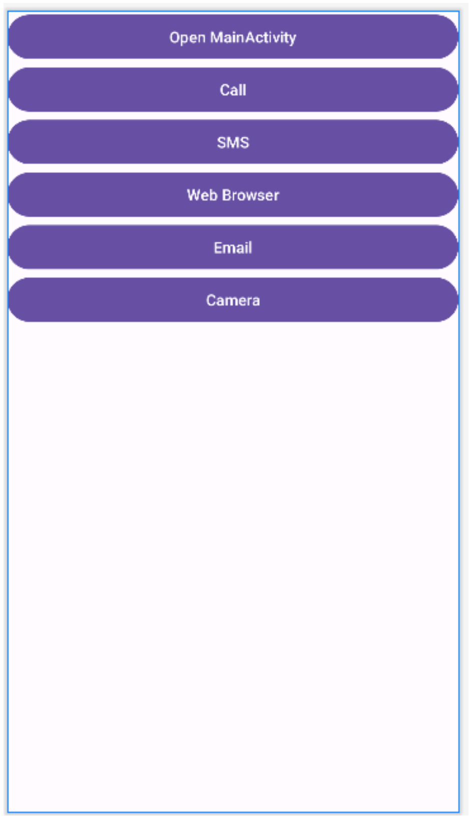
7. Now modify the MainActivity as follows.

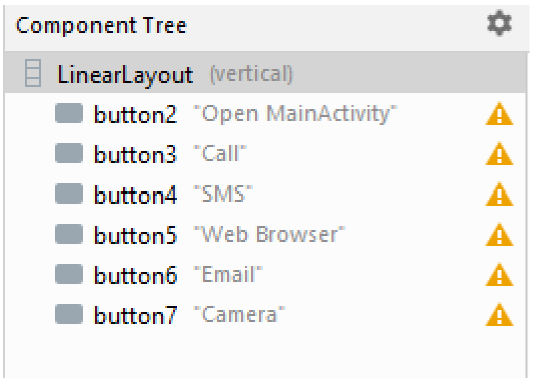
```
class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)      whenever there is a life cycle event
      setContentView(R.layout.activity_main)

      val textView:TextView = findViewById(R.id.textView)
      val button:Button = findViewById(R.id.button)
      val viewModel = ViewModelProvider(this)[MainActivityData::class.java]

      textView.text = viewModel.count.value.toString()
      button.setOnClickListener {
         viewModel.increment()
      }
      viewModel.count.observe(this) {
         textView.text = it.toString()
      }
   }                                   view model- represents the entire life cycle
}
```

8. Now run the program and observe the output by rotating the device or the emulator.

Intents

1. Create a new Activity named HomeActivity.
2. Design the following UI:





Component Tree
LinearLayout (vertical)
button2 "Open MainActivity"
button3 "Call"
button4 "SMS"
button5 "Web Browser"
button6 "Email"
button7 "Camera"

3. Initialize the UIs and implement the button click events.

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_home)
  val button2:Button = findViewById(R.id.button2)
  val button3:Button = findViewById(R.id.button3)
  val button4:Button = findViewById(R.id.button4)
  val button5:Button = findViewById(R.id.button5)
  val button6:Button = findViewById(R.id.button6)
  val button7:Button = findViewById(R.id.button7)

  button2.setOnClickListener {

  }

  button3.setOnClickListener {
    }

    button4.setOnClickListener {

    }

    button5.setOnClickListener {

    }

    button6.setOnClickListener {

    }

    button7.setOnClickListener {

    }
}
```

4. How do we open the HomeActivity as the starting Activity? To do that you need to edit the AndroidManifest as follows:

```
<?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="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MADTutorial07Demo"
        tools:targetApi="31">
        <activity
            android:name=".HomeActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        </activity>
    <activity
        android:name=".MainActivity"
        android:exported="false">

    </activity>
  </application>
</manifest>
```

- **<intent-filter>**

An **<intent-filter>** specifies the types of intents that an activity, service, or broadcast receiver can respond to. It acts as a filter, allowing specific types of intents to be directed to the component (activity/service/receiver) that has this filter. Components that do not have a matching intent filter will not receive the intent.

- **<action android:name="android.intent.action.MAIN" />**

This element defines an action that the component can perform. The value **"android.intent.action.MAIN"** indicates that this is the main entry point to the application. In simpler terms, when the app is launched, this action specifies which activity should be opened first. Usually, it's the activity that represents the app's main screen.

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

This element specifies a category for the intent. The value **"android.intent.category.LAUNCHER"** means that the activity with this intent filter should be listed in the system's app launcher (i.e., the screen where you see all the app icons and can open apps). When you click on the app's icon, this is the activity that gets launched because of the combination of the **MAIN** action and the **LAUNCHER** category.

5. Run the app.
6. Let's implement explicit intent will use button2.

```
button2.setOnClickListener {
  //Explicit Intent to open an activity
  val intent = Intent(this,MainActivity::class.java)
  startActivity(intent)

}
```

7. Now let’s observe how we can use implicit intent to open the dialer from a button click. In this example we are going to pass the number that we want to call to.

```
button3.setOnClickListener {
   //implicit intent to open the dialler with a number
   val number = "+94717123456"
   val uri = Uri.parse(String.format("tel:$number"))
   val intent = Intent()
   intent.action = Intent.ACTION_DIAL
   intent.data = uri
   startActivity(intent)
}
```

8. Next, SMS with the button4.

```
button4.setOnClickListener {
   //implicit intent to send an sms
   val number = "+94717123456"
   val smsText = "Welcome to MAD 2023"
   val uri = Uri.parse(String.format("smsto:$number"))
   val intent = Intent()
   intent.action = Intent.ACTION_SENDTO
   intent.data = uri
   intent.putExtra("sms_body",smsText)
   startActivity(intent)

}
```

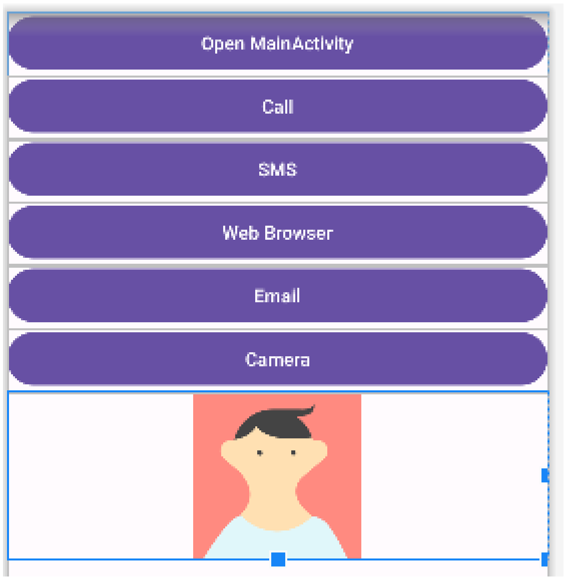
9. Next Web browser for the button 5.

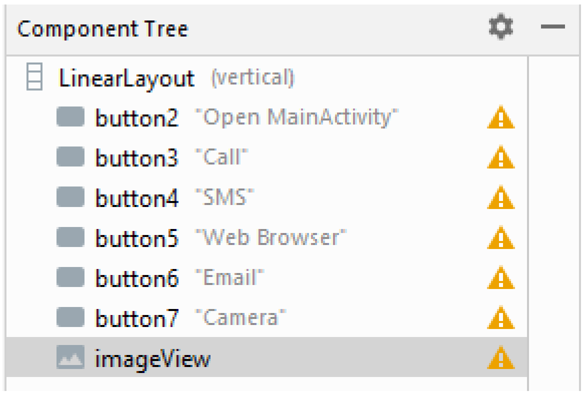
```
button5.setOnClickListener {
   val url = "https://www.sliit.lk/"
   val uri = Uri.parse(url)
   val intent = Intent()
   intent.action = Intent.ACTION_VIEW;
   intent.data = uri
   startActivity(intent)

}
```

10. Button6 for sending emails.

```
button6.setOnClickListener {
   val mailTo = arrayOf("abc@email.com")
   val subject = "Test Email"
   val mailBody = "This the test email body"

   val intent = Intent()
   intent.action = Intent.ACTION_SEND
   intent.type = "message/rfc822"
   intent.putExtra(Intent.EXTRA_EMAIL, mailTo)
   intent.putExtra(Intent.EXTRA_SUBJECT, subject)
   intent.putExtra(Intent.EXTRA_TEXT, mailBody)
   startActivity(intent)
}
```

11. Next, we will use the existing camera app to take an image and show that image on our screen. For that first you need to update the UI to display the image.





```
LinearLayout (vertical)
   button2 "Open MainActivity"
   button3 "Call"
   button4 "SMS"
   button5 "Web Browser"
   button6 "Email"
   button7 "Camera"
   imageView
```

12. Initialize the imageView. Use gloable variable to do that.

```
//Class level
lateinit var imageView: ImageView

//in onCreate()

imageView =findViewById(R.id.imageView)
```

13. For this example, we are going to launch a system app and retrieve data from that app into our application. For this you need to implement object from Activity Launcher on class level as follows:

```
val thumbnailLauncer = registerForActivityResult(
   ActivityResultContracts.StartActivityForResult()
) {
   if (it.resultCode == RESULT_OK){
      val data = it.data
      val imageBitmap = data?.extras?.get("data") as? Bitmap
      imageView.setImageBitmap(imageBitmap)
   }
}
```

14. Now implement the button7 code as follows:

```
button7.setOnClickListener {
   val intent = Intent()
   intent.action = MediaStore.ACTION_IMAGE_CAPTURE
   thumbnailLauncer.launch(intent)
}
```

15. Check the output.

Exercise

Research the following and find out what else you can do with it.

```
registerForActivityResult(
   ActivityResultContracts.StartActivityForResult()
){}
```

https://developer.android.com/training/basics/intents/result