Android Essential

Topics

- Essential Components
- Activity Lifecycle
- Application Architecture: Managing Code & Data
  - ViewModel
  - (Mutable)LiveData
- *Class Exercise: ViewModel + LiveData*

Code Example on GitHub

https://github.com/dulimarta/cs357-code-example

Essential Components

- **Activity / Fragment**: manages user interactions with the UI on the screen
- **Service**
- **Broadcast Receiver**
- **Content Provider (Rarely Used Now)**

Component with UI
Components without UI

Service

- An app component that does not have a user interface
- Runs in the *background*
- Examples
  - Timer
  - Music player
  - Pedometer, Fitness app

Broadcast Receiver

- An app component that handles broadcast messages sent by other Android apps or by the Android system
- Examples
  - Incoming SMS text message/phone calls handler
  - 2FA push notifications: Duo
  - Time zone change
  - Headphone jack (un)plugged
  - Low battery
  - Photo Taken by device
  - *Many more*

Content Provider

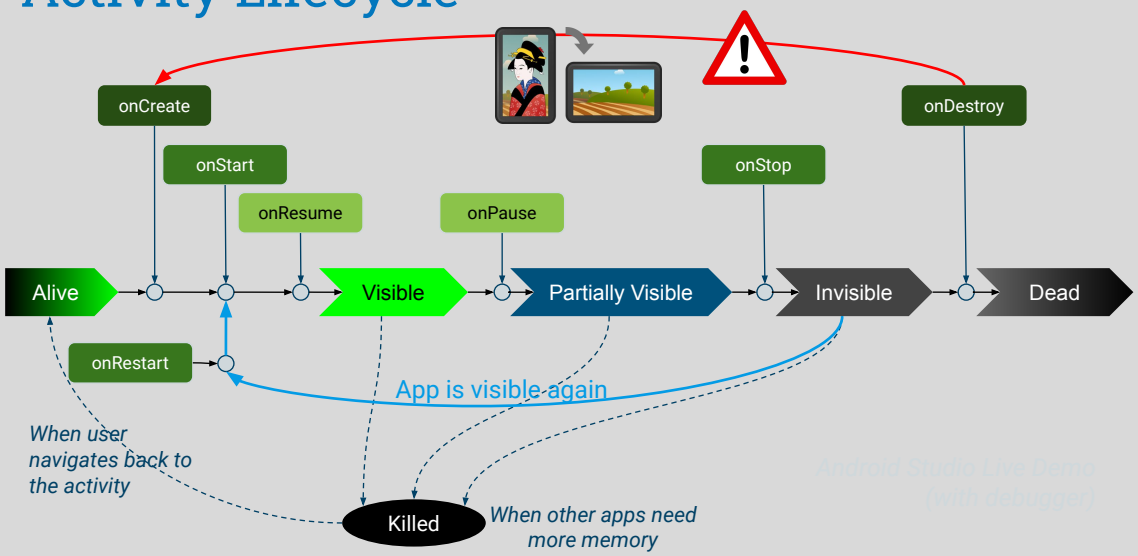
- Allows data of an app to be made available to other apps
- Data can be stored in
  - a SQLite DB
  - Android File system
  - Other local storage
- Examples
  - List of personal contacts
  - Phone call log
- *Fancier functionalities are now provided the the Jetpack Room (Database)*

Activity Lifecycle

Activity Stages of Life

- Alive
- Foreground/interactive, and entirely visible
- Background/non-interactive and partially visible
- Background/non-interactive and completely invisible
- Dead

Activity Lifecycle



Android Studio Live Demo (with debugger)

⚠ Data in Android Activity will NOT survive after ANY configuration changes

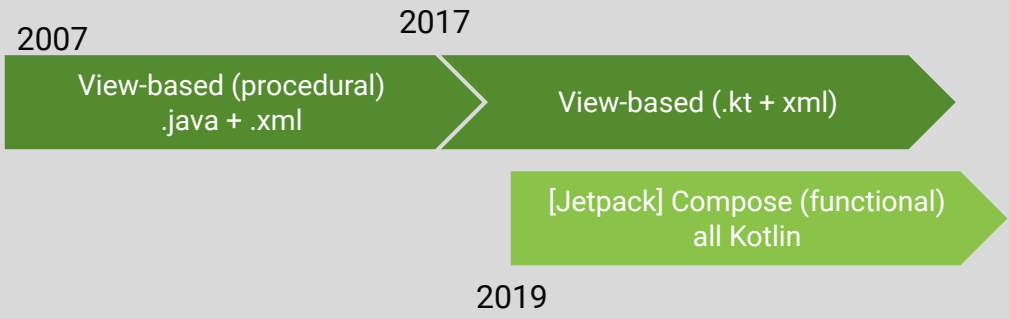
Configuration Changes

- Screen rotation (is just one of the easiest to understand)
- Switching to a different language
- Changing font scale factor
- Switching to a different font

UI Layout & Widgets

iOS	Android
UIViewController	Activity / Fragment
XIB, NIB	Layout
Storyboard	Navigation graph
IBOutlet	View reference
IBAction	Event listener
UILabel	TextView
UIButton	Button
UITextField	TextEdit
UITableView	RecyclerView

Android User Interface Design Paradigms



Activity, Layout, View Refs (View Based)

```
// MyActivity.kt
class MyActivity: AppCompatActivity() {
  override fun onCreate(____) {
    super.onCreate(____)z
    setContentView(R.layout.my_layout)
    val hello = findViewById<TextView>(R.id.my_label)
    val go = findViewById<Button>(R.id.go_btn)
    go.setOnClickListener {
      hello.text = “Hello Android World”
    }
  }
}
```

```
// res/layout/my_layout.xml
<ConstraintLayout>
  <TextView
    android:id=”@+id/my_label”
    android:text=”Hello World!” />
  <Button
    android:id=”@+id/go_btn” />
</ConstraintLayout>
```

Kotlin for app/UI logic

XML for UI structure

Activity, Composable UI Functions

```
// MyActivity.kt (View based)
class MyActivity: AppCompatActivity() {
  override fun onCreate(____) {
    super.onCreate(____)z
    setContentView(R.layout.my_layout)
    val hello = findViewById<TextView>(R.id.my_label)
    val go = findViewById<Button>(R.id.go_btn)
    go.setOnClickListener {
      hello.text = “Hello Android World”
    }
  }
}
```

```
// res/layout/my_layout.xml
<ConstraintLayout>
  <TextView
    android:id=”@+id/my_label”
    android:text=”Hello World!” />
  <Button
    android:id=”@+id/go_btn” />
</ConstraintLayout>
```

```
// MyActivity.kt (Jetpack Compose)
class MyActivity: ComponentActivity() {
  override fun onCreate(____) {
    super.onCreate(____)
    setContentView {
      MyUI()
    }
  }
}

@Composable
fun MyUI() {
  Row {
    Text(“Hello World”)
    Button(onClick = {    }) {
      Text(“Go”)
    }
  }
}
```

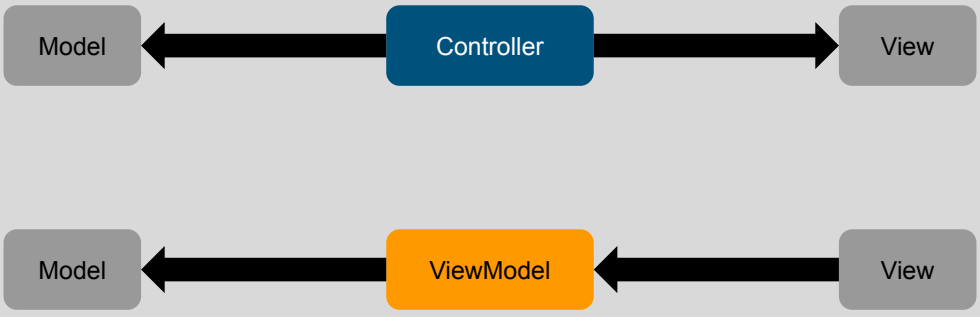
Application Architecture

Android Architecture Components

MVVM

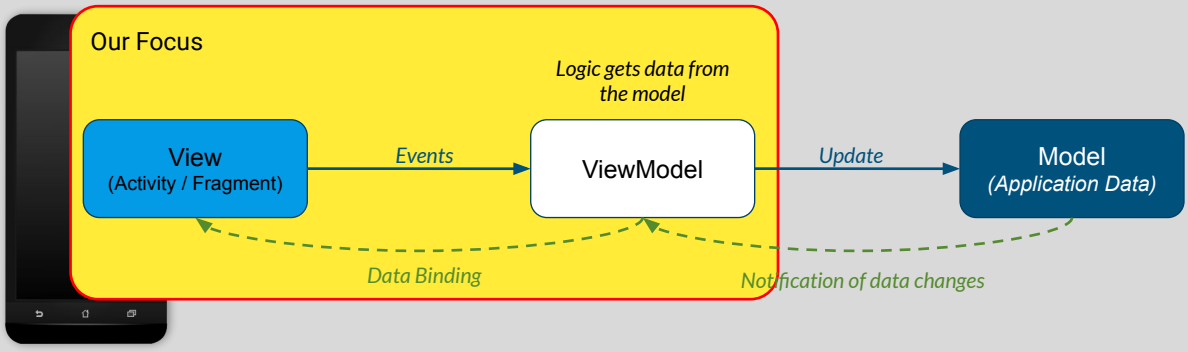
- **Model**
- **View**
- **ViewModel**

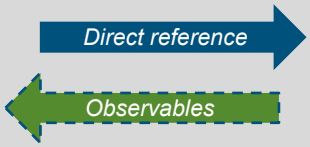
iOS MVC vs. Android MVVM



Uni-Directional Flow

Intercommunication in MVVM





MVC in iOS

Role	Implementation File	Description
Model	`.swift`	Holds application data
View	`.storyboard or .xib`	UI objects that renders on screen
Controller	`.swift`	Provide data for view, handle interactions with view

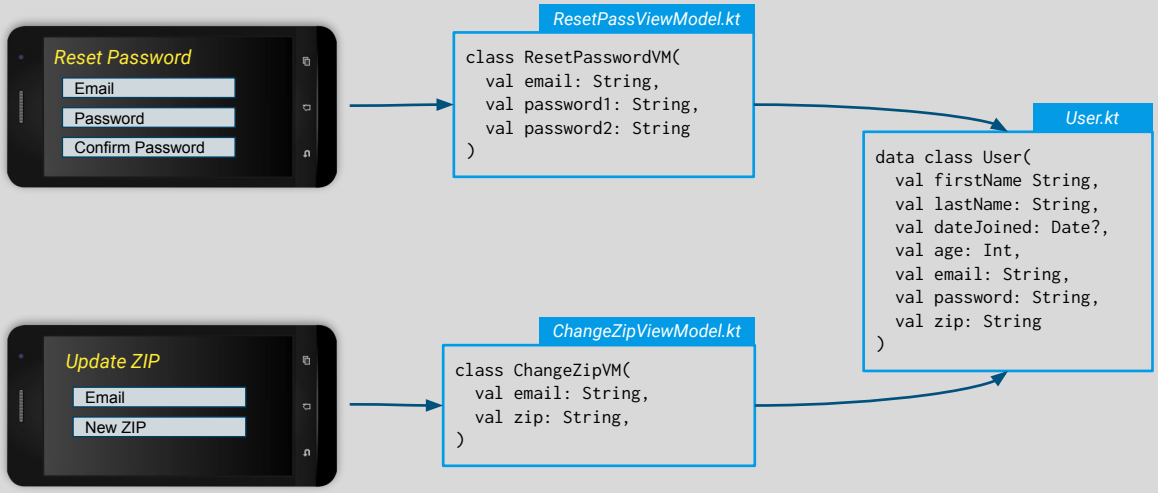
MVVM in Android

Role	Implementation	Description
Model	`.kt`	Holds application data
View	`Activity or Fragment or @Composable`	Handle *only* what the user sees or touches on the screen (no business logic)
ViewModel	`.kt`	Provides data for the view, handles UI logic

What is ViewModel

- A LifeCycle aware object which knows how to save/retain/restore data across Activity lifecycle changes
- A ViewModel is a (sub)model designed to provide backing store for a specific Activity
  - As opposed to the Model itself which is designed to hold data of the entire app

MVVM Example (Simplified)



ViewModel: “model” for a specific view

Model: model for the app

LiveData

- Observable data holder
  - Holder: an object that holds the data of your app
  - Observable: changes to your data can be observed by the view
- Two variations:
  - `LiveData<T>`
  - `MutableLiveData<T>`
- Updating (mutable) livedata
  - Use `setValue()` if called from the main thread
  - Use `postvalue()` if called from a background thread
- Best practices
  - Do not expose `MutableLiveData` for public access, keep them `private`
  - Only allow mutation of your (live) data via public functions

Android ViewModel + LiveData

```
// (a) Basic skeleton
class MainActivityViewModel: ViewModel() {
   var counter = 73
   fun addCounter() {
     counter = counter + 1
   }
}
```

```
// (b) Use LiveData
class MainActivityViewModel: ViewModel() {
   val counter = MutableLiveData<Int>(73)

   fun addCounter() {
      counter.value = counter.value!! + 1
   }
}
```

```
// (c) hide mutable livedata from public access
class MainActivityViewModel: ViewModel() {
   private val _counter = MutableLiveData<Int>(73)
   val counter: LiveData<Int> get() {
     return _counter
   }

   fun addCounter() {
      _counter.value = _counter.value!! + 1
   }
}
```

- *The mutable is hidden from public*
- *Public can only read (get) the content*

Android ViewModel + LiveData

```
class MyActivity: AppCompatActivity() {
  lateinit var myViewModel: MainActivityViewModel
  var lateinit label: TextView

  override fun onCreate(savedInstanceState: Bundle!) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.my_activity)
    label = findViewById<TextView>(R.id.label)
    val addBtn = findViewById<Button>(R.id.add_button)
    addBtn.setOnClickListener {
       myViewModel.addCounter()


       label.text = myViewModel.counter.toString()
    }
  }

  override fun onResume() {
    label.text = myViewModel.counter.toString()
  }
}
```

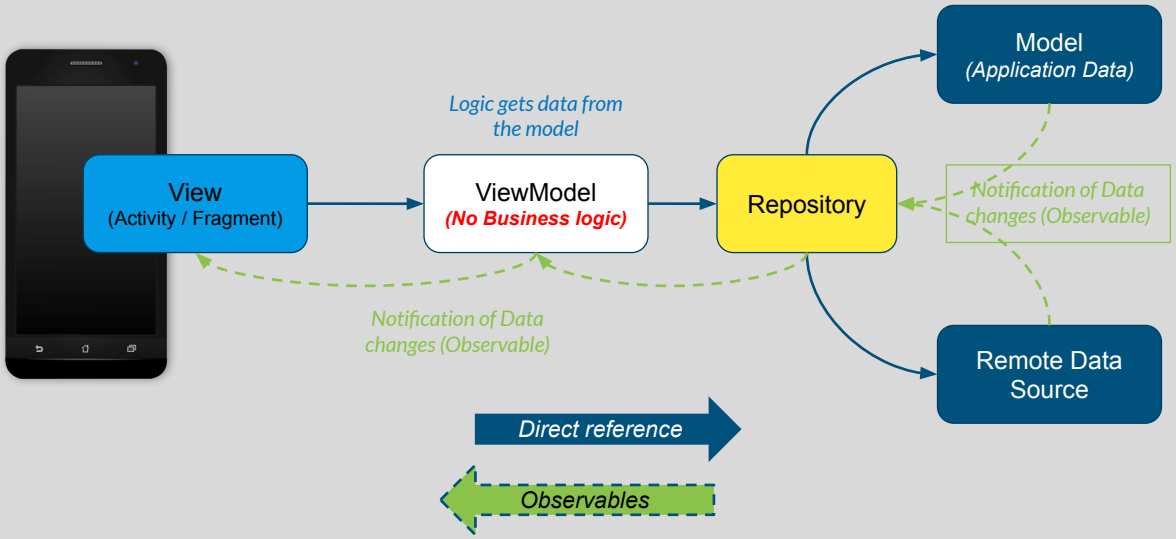
Without LiveData

```
class MyActivity: AppCompatActivity() {
  lateinit var myViewModel: MainActivityViewModel


  override fun onCreate(savedInstanceState: Bundle!) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.my_activity)
    val label = findViewById<TextView>(R.id.label)
    val addBtn = findViewById<Button>(R.id.add_button)
    addBtn.setOnClickListener {
       myViewModel.addCounter()
    }
    myviewModel.counter.observe(this) {
       label.text = it.toString()
    }
  }

  override fun onResume() {
    // No code needed here
  }
}
```

With LiveData

Demo: app-arch

MVVM for Larger Apps



Reading Assignment

Engelsma/Dulimarta textbook

- Chapters 3 & 4

ViewModelFactory

Motivation

- The ViewModel examples used so far have no dependencies
- Sometimes, our ViewModel requires additional external dependencies
- Practical use cases
  - Initialization logic requires a token string to connect to remote server
  - Bigger apps usually require multiple VIewModels which share a common repository object
- Solution: use a *factory design pattern* [1] [2] to construct viewmodels

ViewModel + Factory

```
// YourViewModel.kt
import androidx.lifecycle.*
class MainActivityViewModel(val countryCode: String): ViewModel() {
   init {
     // code that depends on countryCode
   }
}

class VMFactory(val code:String): ViewModelProvider.NewInstanceFactory() {
   override fun <T : ViewModel> create(modelClass: Class<T>): T {
       return MainActivityViewModel(code) as T
   }
}
```

```
// YourActivity.kt
import androidx.activity.viewModels
class YourActivity: AppCompatActivity() {
  val myViewModel by viewModels<MainActivityViewModel>() {
    VMFactory(“UK”)
  }
}
```