# CMSC436: Programming Handheld Systems

# Lifecycle-Aware Components

# Today’s Topics

Lifecycle-Aware Components

ViewModel

Live Data

# Android App Behavior

Multiple entry points launched individually

Components started in many different orders

Android kills components on reconfiguration / low memory

# Key Architectural Goals

Don’t store app data or state in your app components

Don’t design your app components so they depend on each other

# Lifecycle-Aware Components

Links app components to their lifecycle events

LifeCycle – Represents Android lifecycle

LifecycleOwner – A component with an Android lifecycle

LifecycleObserver – Callbacks for listening to lifecycle changes

# Lifecycle

Holds information about the lifecycle state of an Android component

State – Enum representing lifecyle states

Events – Enum representing lifecycle events (transitions between states)

# Lifecycle.State

INITIALIZED - Initialized state for LifecycleOwner

CREATED - Created state for LifecycleOwner

DESTROYED - Destroyed state for LifecycleOwner

RESUMED - Resumed state for LifecycleOwner

STARTED - Started state for LifecycleOwner

# Lifecycle.Event

ON_ANY - Constant matching all events

ON_CREATE - onCreate event of the LifecycleOwner

ON_DESTROY - onDestroy event of the LifecycleOwner

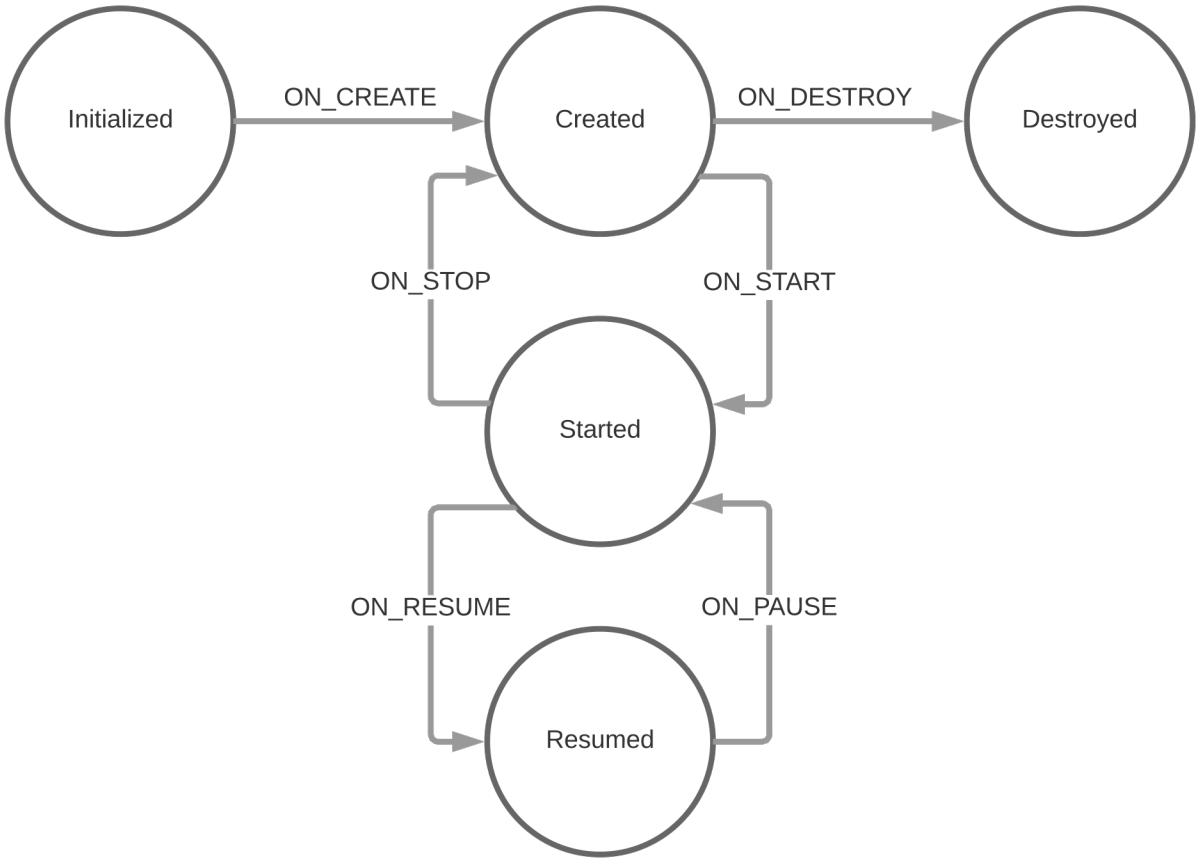
ON_PAUSE - onPause event of the LifecycleOwner

ON_RESUME - onResume event of the LifecycleOwner

ON_START - onStart event of the LifecycleOwner

ON_STOP - onStop event of the LifecycleOwner

# Lifecycle State Model



# Lifecycle Methods

void addObserver(LifecycleObserver observer)

Adds a LifecycleObserver that will be notified when the LifecycleOwner changes state

void removeObserver(LifecycleObserver observer)

Removes the given observer from the observers list

Lifecycle.State getCurrentState()

Returns the current state of the Lifecycle

# LifecycleOwner

Represents a component with an Android lifecycle

An interface that returns a Lifecycle object from the getLifecycle() method

# LifecycleObserver

- Callbacks for listening to lifecycle changes to a LifecycleOwner
- Our examples will use Java 8
  - Observe events with DefaultLifecycleObserver
  - Add "android.arch.lifecycle:common-java8:<version>" to module’s build.gradle file

# DefaultLifecycleObserver Methods

void onCreate(LifecycleOwner owner)

Notifies that ON_CREATE event occurred.

void onStart(LifecycleOwner owner)

Notifies that ON_START event occurred.

void onResume(LifecycleOwner owner)

Notifies that ON_RESUME event occurred.

# DefaultLifecycleObserver Methods

void onDestroy(LifecycleOwner owner)

Notifies that ON_DESTROY event occurred

void onPause(LifecycleOwner owner)

Notifies that ON_PAUSE event occurred

void onStop(LifecycleOwner owner)

Notifies that ON_STOP event occurred

# LifecycleObserver Methods

ON_CREATE, ON_START, ON_RESUME events are dispatched after the LifecycleOwner's related method returns

ON_PAUSE, ON_STOP, ON_DESTROY events are dispatched before the LifecycleOwner's related method is called

# Today’s Topics

Lifecycle-Aware Components

ViewModel

Live Data

# ViewModel Responsibilities

Responsible for managing data for an Activity or a Fragment (owner)

Handles communication between the Activity or Fragment and the rest of the application

# ViewModel Lifecycle

- Associated with a scope (e.g., a Fragment or an Activity)
- Retained as long as the scope is alive
  - Will not be destroyed if its owner is destroyed for a configuration change
  - The new instance of the owner will be reconnected to the existing ViewModel

# ViewModel Implementation Rules

Should never access the View hierarchy or hold a reference to the Activity or the Fragment

# ViewModel Methods

void onCleared()

This method will be called when this ViewModel is no longer used and will be destroyed

# ViewModelProvider

static ViewModelProvider
of (FragmentActivity activity)

Creates a ViewModelProvider, which retains ViewModels while a scope of given Activity is alive

# Today’s Topics

Lifecycle-Aware Components

ViewModel

Live Data

# LiveData

- Data holder observable within a given lifecycle
  - Observer paired with a LifecycleOwner
  - Observer notified when data changes, only if the LifecycleOwner is in active state
    - LifecycleOwner is considered active, if its state is STARTED or RESUMED
- Designed to hold individual data fields of ViewModel
- Can also be used to share data between components



LifecycleAware Ticker

# TickerDisplayActivity,kt

```
override fun onCreate(savedInstanceState: Bundle?) {
        …
        mCounterView = findViewById(R.id.counter)
        // Get reference to TickerViewModel
        mTickerViewModel =
         ViewModelProviders.of(this).get(TickerViewModel::class.java)
        // Display initial Ticker value
        mCounterView.text = mTickerViewModel.counter.value.toString()
        // Observe changes to Ticker
        beginObservingTicker()
        // Tie TickerViewModel to Activity lifecycle
        mTickerViewModel.bindToActivityLifecycle(this)
    }
```

# TickerDisplayActivity,kt

```
private fun beginObservingTicker() {
    // Create Observer
    val tickerObserver =
            Observer<Int> {mCounterView.text = it.toString()}
    // Register observer
    mTickerViewModel.counter.observe(this, tickerObserver)
}
```

# TickerViewModel.kt

```
private val mCounter = MutableLiveData<Int>()
…
internal val counter: LiveData<Int>
    get() = mCounter

private val updater: Runnable = object : Runnable {
    override fun run() {
        mCounter.value = mCounter.value!! + 1
        mHandler.postDelayed(this, ONE_SECOND.toLong())
    }
}
init {
    // Set initial value
    mCounter.value = 0
    mHandler = Handler()
}
```

# TickerViewModel.kt

```
internal fun bindToActivityLifecycle(
                            tickerDisplayActivity: TickerDisplayActivity) {
    tickerDisplayActivity.lifecycle.addObserver(this)
}

override fun onResume(owner: LifecycleOwner) {
    // Update the elapsed time every second.
    mHandler.postDelayed(updater, 1000L)
}

override fun onPause(owner: LifecycleOwner) {
    // Cancel work on Handler
    mHandler.removeCallbacks(updater)
}
```

# Best Practices

Keep your UI controllers (activities and fragments) as lean as possible. They should not try to acquire their own data; instead, use a ViewModel to do that, and observe the LiveData to reflect the changes back to the views

Try to write data-driven UIs where your UI controller’s responsibility is to update the views as data changes, or notify user actions back to theViewModel

Put your data logic in your ViewModel class. ViewModel should serve as the connector between your UI controller and the rest of your application

Never reference a View or Activity context in your ViewModel. If the ViewModel outlives the activity (in case of configuration changes), your activity will be leaked and not properly garbage-collected

# Next Time

Firebase

# Example Applications

LifecycleAwareTicker