Lesson 7:

Unscrambler

Instructor: Ahmet Geymen

About this lesson

- Lesson 7:
  - Activity Lifecycle
  - Logging
  - Architectural Principles
    - Unidirectional Data Flow
    - ViewModel
- Workshop
  - Dessert Clicker
  - Unscrambler

Get started

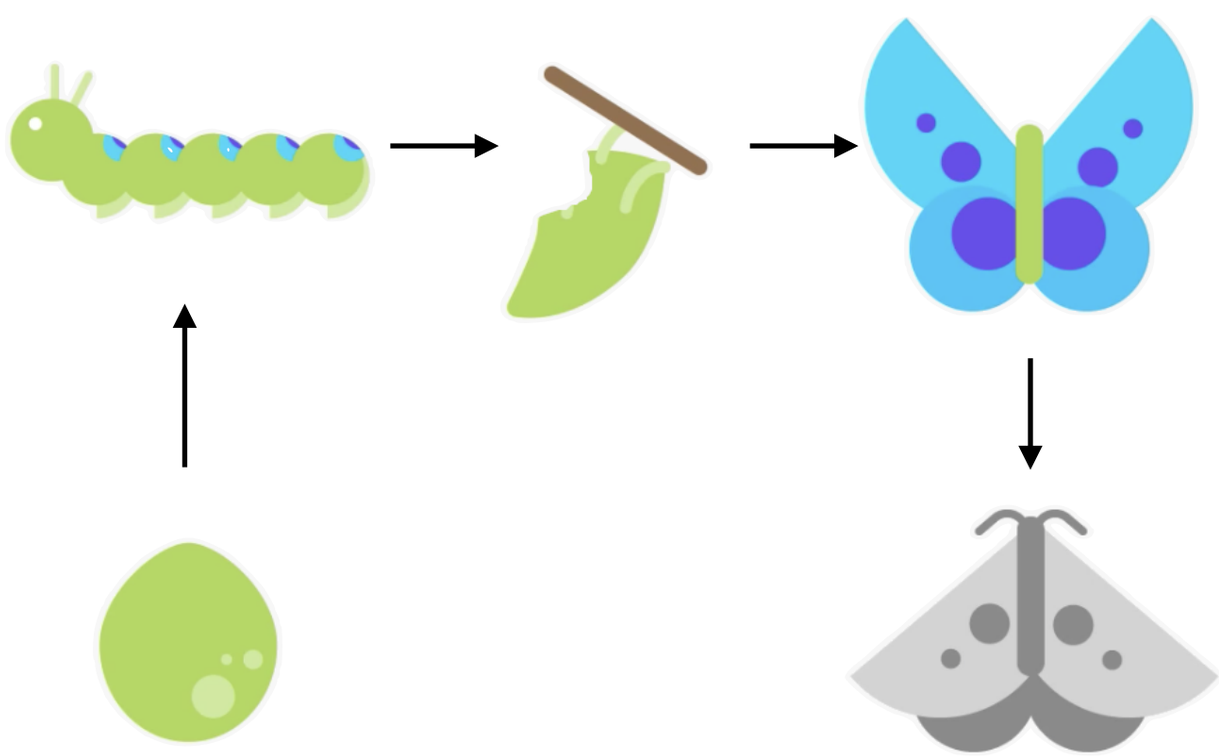
Activity lifecycle & App Architecture

Activity Lifecycle

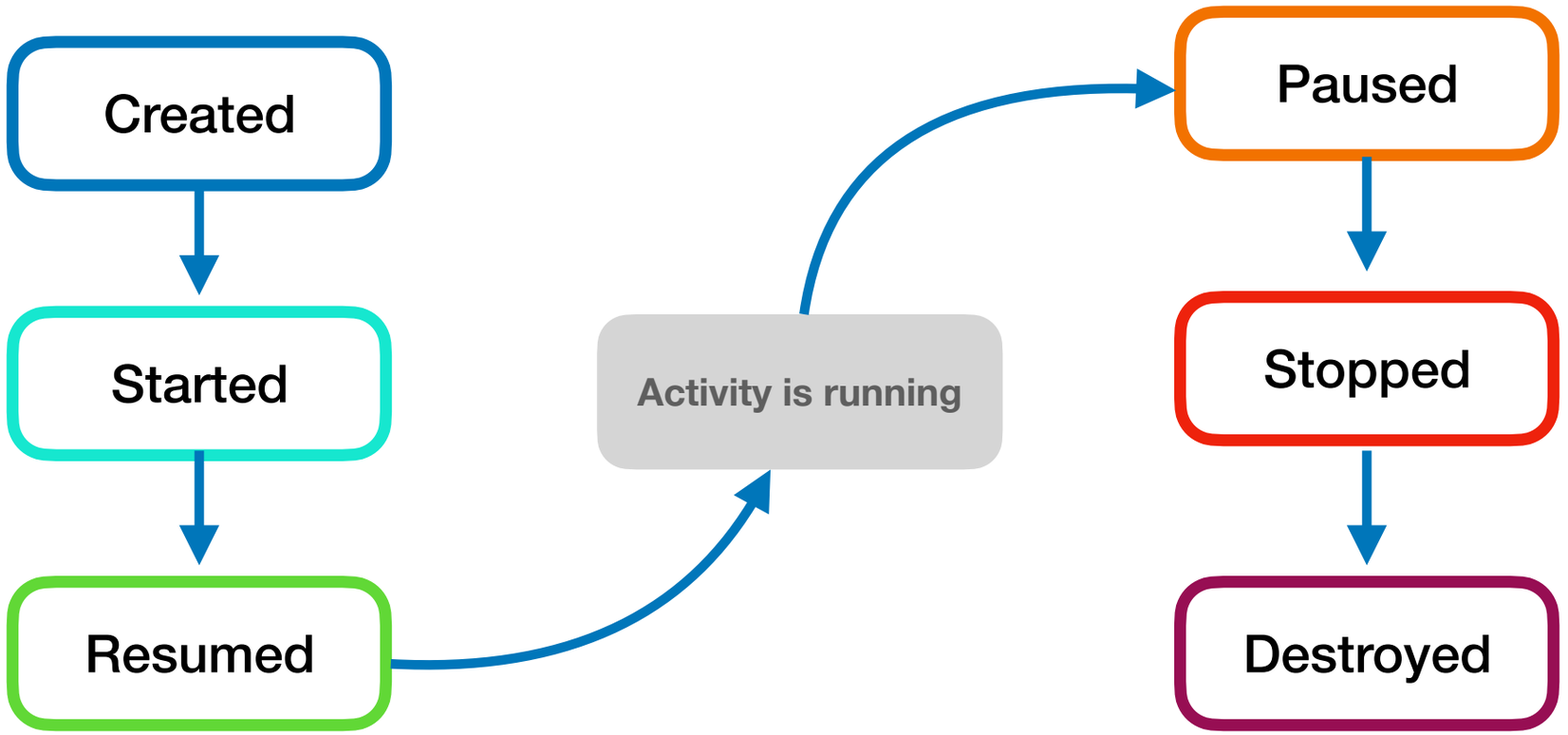
Why it matters

- Preserve user data and state if:
  - User temporarily leaves app and then returns
  - User is interrupted (for example, a phone call)
  - User rotates device
- Avoid memory leaks and app crashes

Simplified activity lifecycle

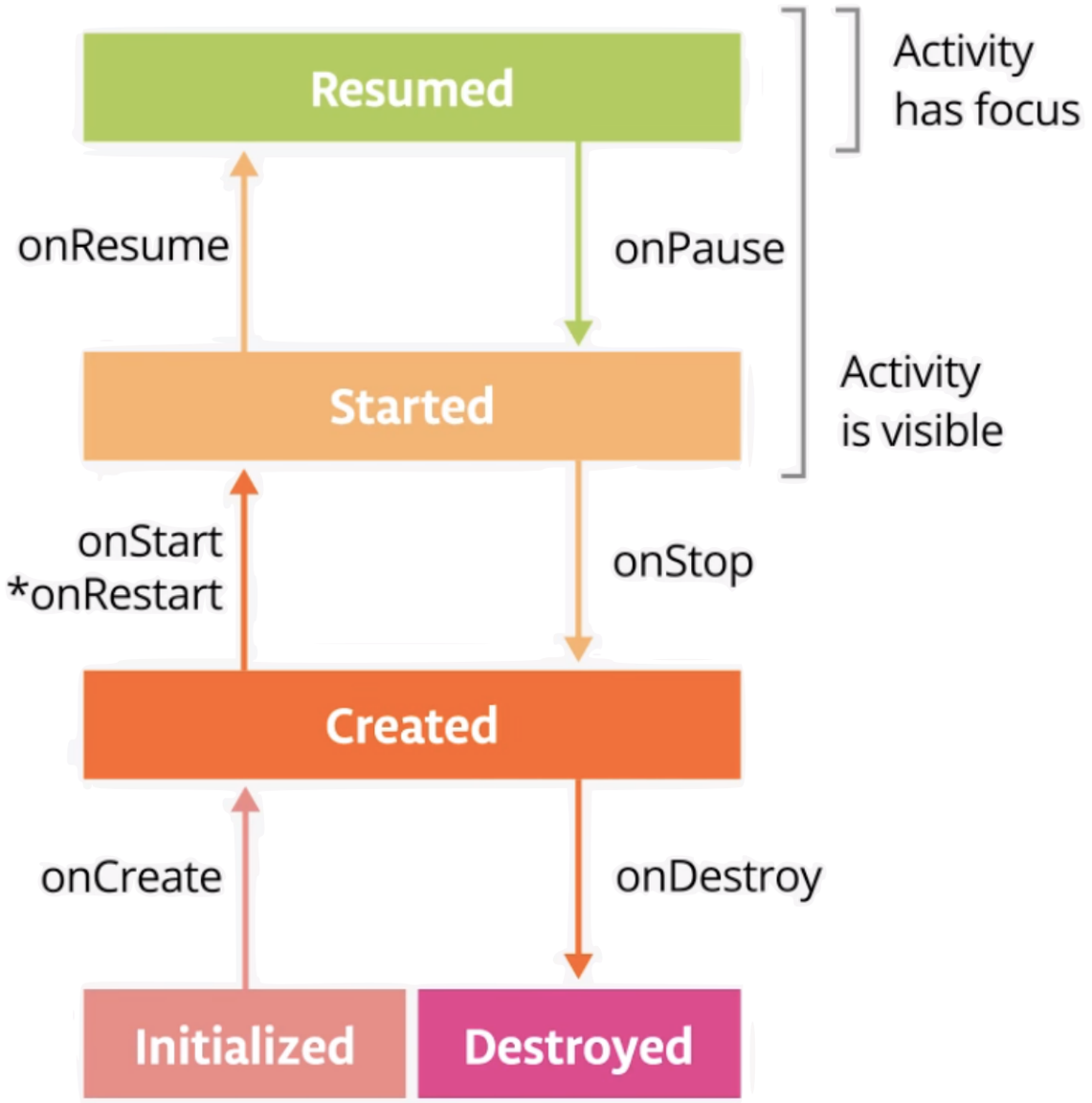


Simplified activity lifecycle



Activity Lifecycle

Activity Lifecycle



onCreate()

- Activity is created and other initialization work occurs
- You must implement this callback
- Inflate activity UI and perform other app startup logic

onStart()

- Activity becomes visible to the user
- Called after activity:
  - `onCreate()` or
  - `onRestart()` if activity was previously stopped

onResume()

- Activity gains input focus:
  - User can interact with the activity
- Activity stays in resumed state until system triggers activity to be paused

onPause()

- Activity has lost focus (not in foreground)
- Activity is still visible, but user is not actively interacting with it
- Counterpart to `onResume()`

onStop()

- Activity is no longer visible to the user
- Release resources that aren’t needed anymore
- Save any persistent state that the user is in the process of editing so they don’t lose their work

onDestroy()

- Activity is about to be destroyed, which can be caused by:
  - Activity has finished or been dismissed by the user
  - Configuration change
- Perform any final cleanup of resources.
- Don’t rely on this method to save user data (do that earlier)

Summary of activity states

State	Callbacks	Description
Created	`onCreate()`	Activity is being initialized.
Started	`onStart()`	Activity is visible to the user.
Resumed	`onResume()`	Activity has input focus.
Paused	`onPause()`	Activity does not have input focus.
Stopped	`onStop()`	Activity is no longer visible.
Destroyed	`onDestroy()`	Activity is destroyed.

Logging

Logging in Android

- Monitor the flow of events or state of your app.
- Use the built-in `Log` class or third-party library.
- Example `Log` method call: `Log.d(TAG, "Message")`

Write logs with different levels

Priority level	Log method
Verbose	`Log.v(String, String)`
Debug	`Log.d(String, String)`
Info	`Log.i(String, String)`
Warning	`Log.w(String, String)`
Error	`Log.e(String, String)`

Architectural Principles

Why you need good app architecture

- Clearly defines where specific business logic belongs
- Makes it easier for developers to collaborate
- Makes your code easier to test
- Lets you benefit from already-solved problems
- Saves time and reduces technical debt as you extend your app

Common architectural principles

- Separation of concerns
- Driving UI from data models
- Single source of truth
- Unidirectional Data Flow
- Layered architecture

Separation of concerns

- A design principle for separating a computer program into distinct sections.
- A design principle states that the app is divided into classes of functions, each with separate responsibilities.
- Modularity is achieved by encapsulating information inside a section of code that has a well-defined interface.

Drive UI from a model

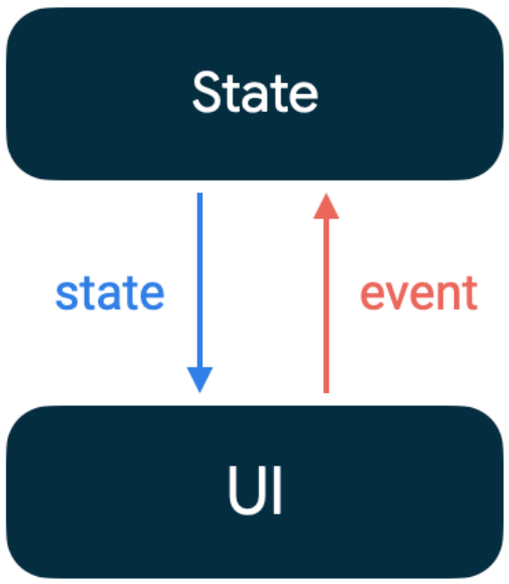
- A principle states that you should drive your UI from a model, preferably a persistent model.
- Data models represent the data of an app. They're independent from the UI elements and other components in your app.
- Not tied to the UI and app component lifecycle.

Single source of truth

- The SSOT is the owner of that data, and only the SSOT can modify or mutate it.
- This pattern brings multiple benefits:
  - It centralizes all the changes to a particular type of data in one place.
  - It protects the data so that other types cannot tamper with it.
  - It makes changes to the data more traceable. Thus, bugs are easier to spot.

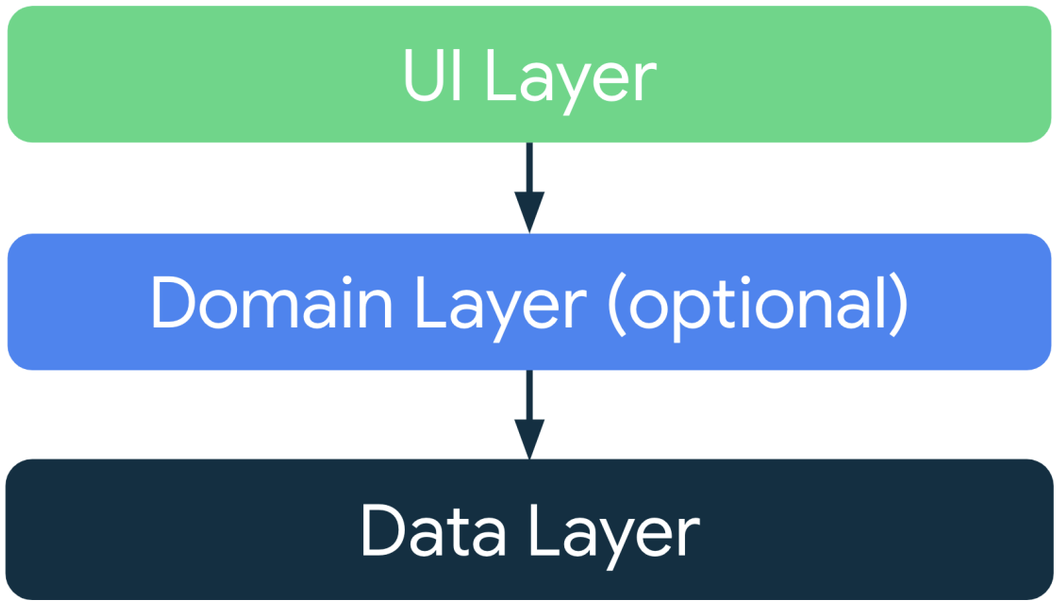
Unidirectional Data Flow

- A design pattern where state flows in only one direction. The events that modify the data flow in the opposite direction.
- By following UDF, you can decouple composables that display state in the UI from the parts of your app that store and change state.



Layered architecture

- The UI layer that displays application data on the screen.
- The data layer that contains the business logic of your app and exposes application data.



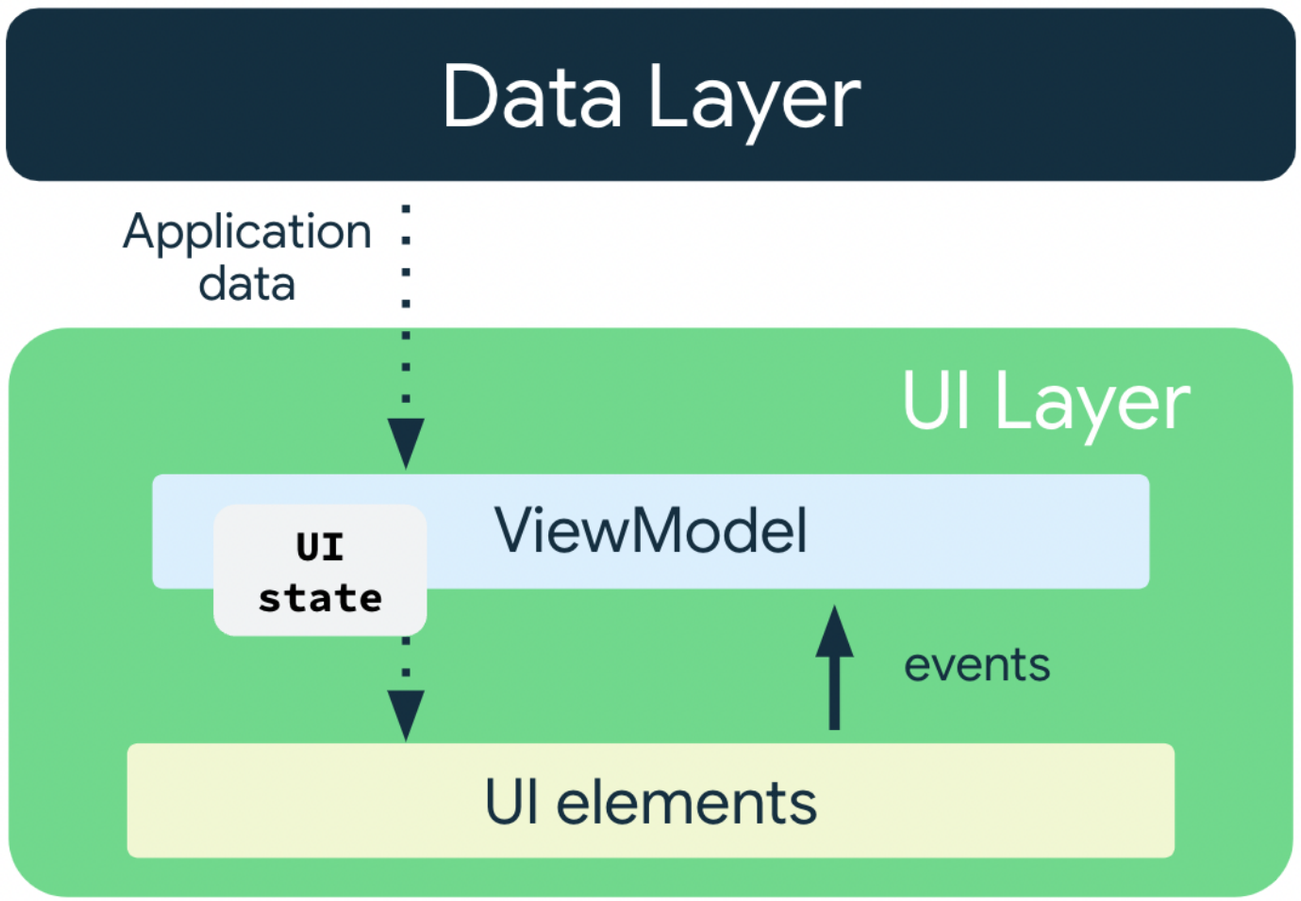
Layered architecture

Layer	Function	Examples
UI	• User interactions • OS interactions • App layout and screens	• Text • Images • Buttons/onClick behavior • Text edit fields/user input
Data	• Information in the app	• Email title, data, subject, body (email app) • Article title, contents, images (news app)

UI Layer

- The role of the UI layer (or presentation layer) is to display the application data on the screen
    - UI elements that render the data on the screen. You build these elements using Views or Jetpack Compose functions.
    - State holders (such as ViewModel classes) that hold data, expose it to the UI, and handle logic.

UI Layer


Data Layer
Application data
UI Layer
ViewModel
UI state
events
UI elements

ViewModel

- Holds and exposes the state that UI consumes
- ViewModel lets your app follow the architecture principle of driving the UI from the model
- Enables data to survive configuration changes
- Stores the app-related data that isn't destroyed when the activity is destroyed and recreated by the Android framework

Workshop