Android Development

Bas Suphakrit Nakhabat

Topic

- Tools and project structure
- View Layout
- Basic Kotlin
- View (Activity, Fragment)
- Android Jetpack
- Network (OkHttp, Retrofit, Gson)
- Kotlin Flows

Tools and project structure

android studio

Android Studio provides the fastest tools for building apps on every type of Android device.

https://developer.android.com/studio

Tools: Gradle

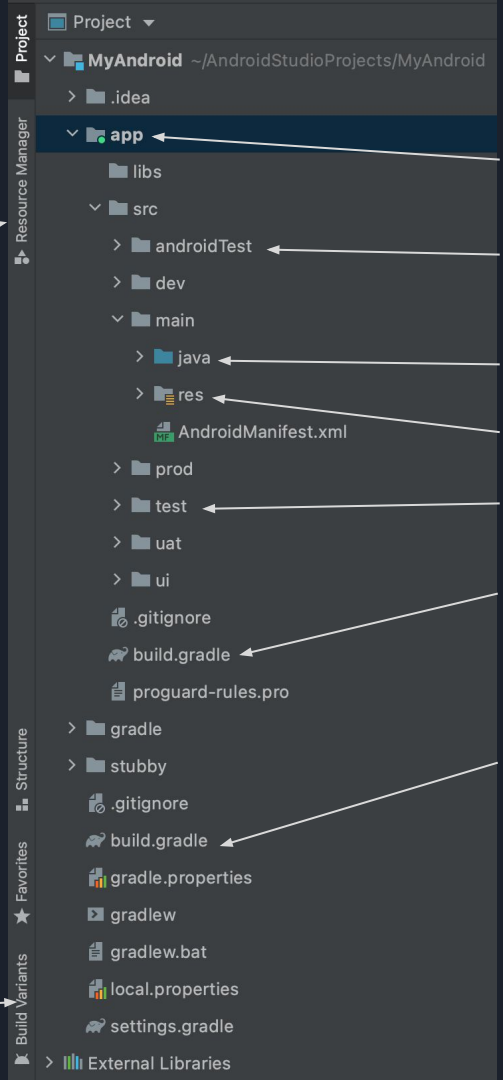
Gradle is an open-source build automation tool focused on flexibility and performance. Gradle build scripts are written using a Groovy or Kotlin DSL.

https://gradle.org/

Project structure

Resource Manager

ENV



Module app

UI Test, Integration Test

Source code

Resource

Unit Test

Build configuration (app module)

Build configuration (Project)

MyAndroid › app › src › main › java › com › example › myandroid › ui › MainActivity

APP | XIAOMI MI A1

Project

MyAndroid ~/AndroidStudioProjects/MyAndroid

- .gradle
- .idea
- app
  - build
  - libs
  - src
    - androidTest
    - dev
    - main
      - java
      - res
      - AndroidManifest.xml
    - prod
    - test
    - uat
    - ui
  - .gitignore
  - build.gradle
  - proguard-rules.pro
- build
- gradle
- stubby
- .gitignore
- build.gradle
- gradle.properties
- gradlew
- gradlew.bat

MainActivity.kt

```
1  package com.example.myandroid.ui
2
3  import ...
10
11 class MainActivity : BaseActivity() {
12
13     private val binding by lazy { ActivityMainBinding.inflate(layoutInflater) }
14     private val viewModel: MainViewModel by viewModel()
15
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         setContentView(binding.root)
19         initObserver()
20     }
21
22     private fun initObserver() {
23         val owner = this
24         with(viewModel) {  this: MainViewModel
25             navigateToActivityNavigation.observe(owner, {
26                 navigateToActivityNavigation()
27             })
28             navigateToFragmentNavigation.observe(owner, {
29                 navigateToFragmentNavigation()
30             })
31             navigateToCallService.observe(owner, {
32                 navigateToCallService()
33             })
34         }
35     }
36
37     private fun navigateToActivityNavigation() {
38         FirstActivity.startIntent( context: this)
```

TODO | Problems | Terminal | Logcat | Profiler | Debug | Run | Find | Build | App Inspection | Event Log | Layout Inspector

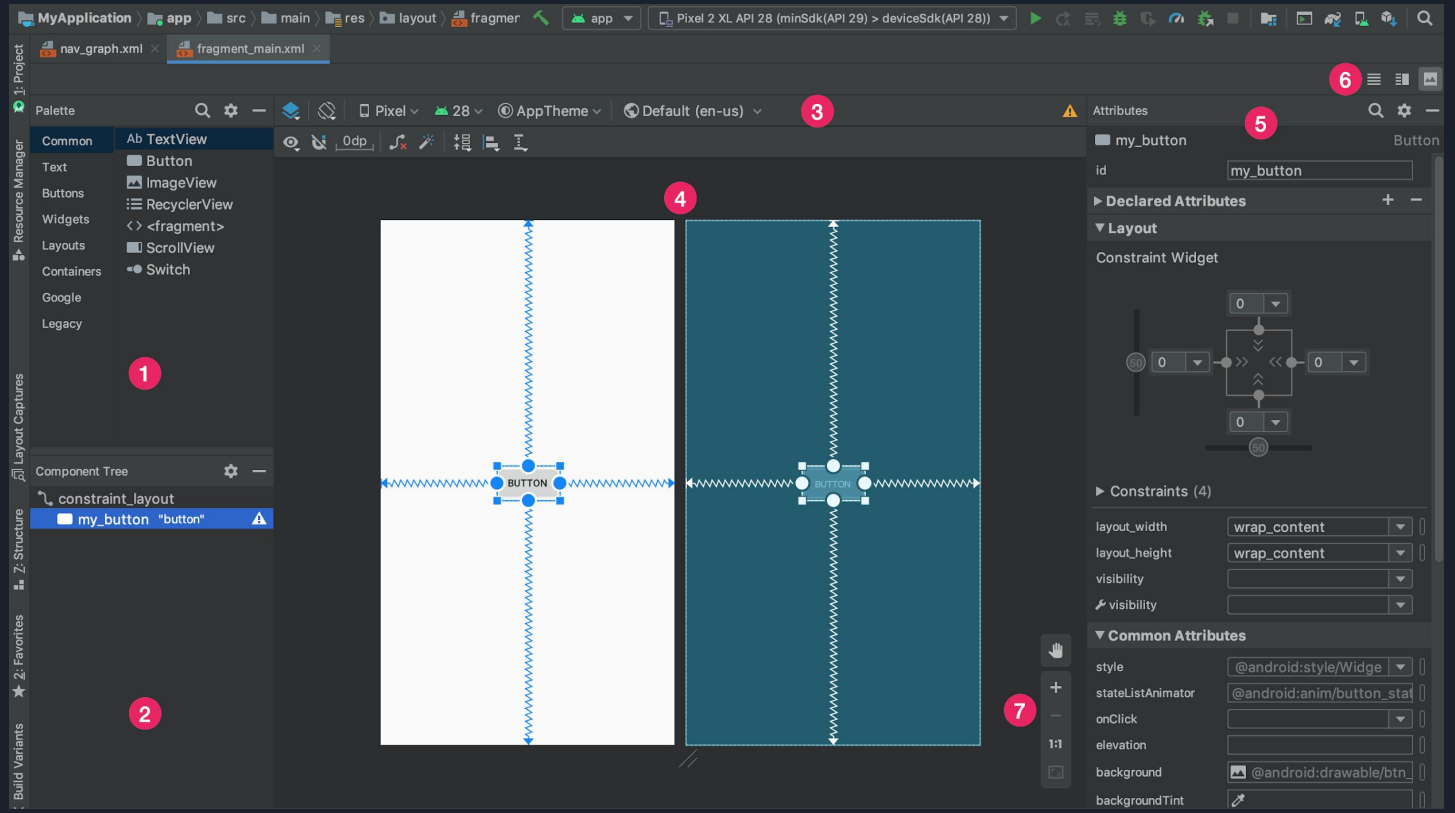
21:1 LF UTF-8 4 spaces Darcula

Resources

```
MyProject/
    src/...
        MyActivity.kt
    res/
        anim/
            fade_in.xml
            fade_out.xml
        color/
            text_color_state.xml
        drawable/
            graphic.png
        font/
            font_kanit.ttf
        layout/
            activity_main.xml
            fragment_info.xml
        mipmap/
            icon_launcher.png
```

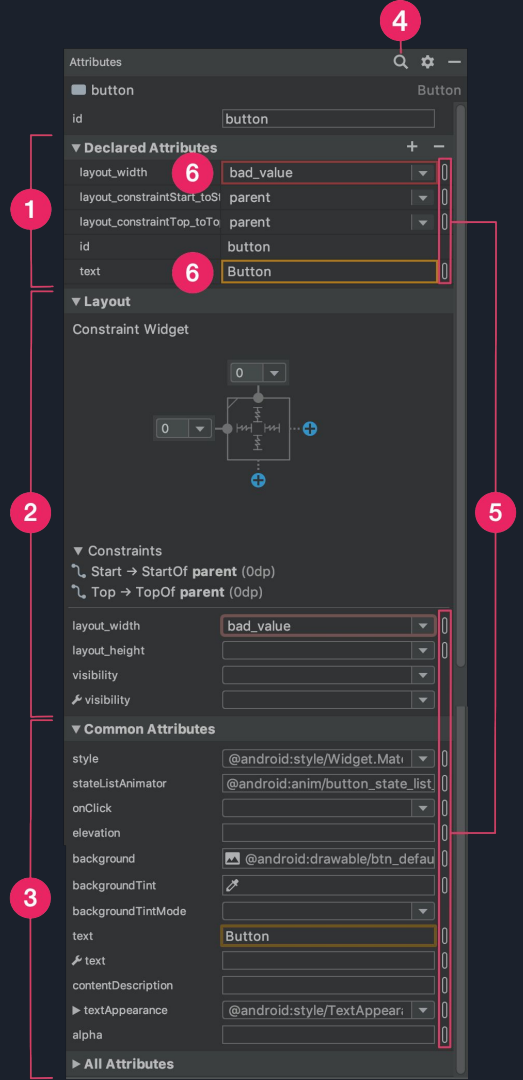
```
MyProject/
    src/...
        MyActivity.kt
    res/
        ...
        values/
            dimens.xml
            strings.xml
            styles.xml
            colors.xml
        values-th/
            strings.xml
```

Layout: Editor

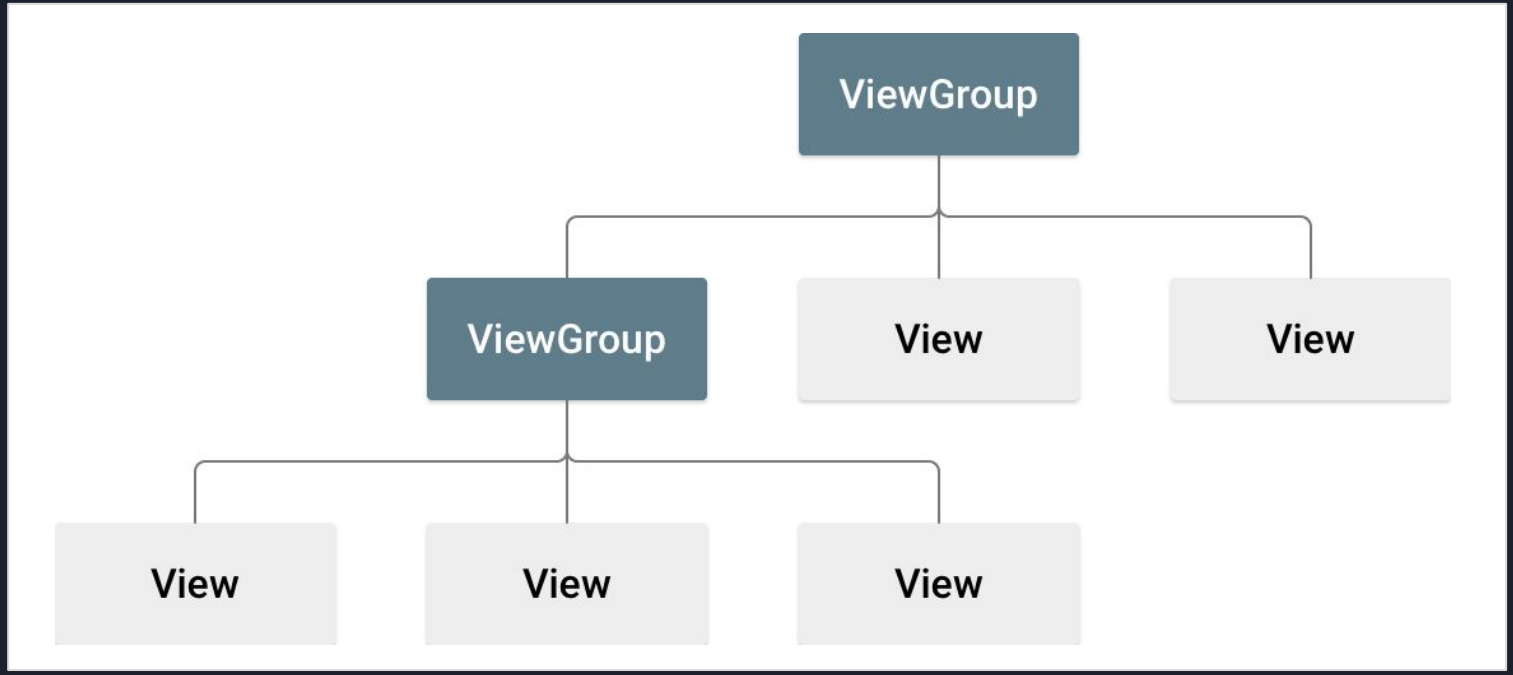


https://developer.android.com/studio/write/layout-editor

Layout: Editor

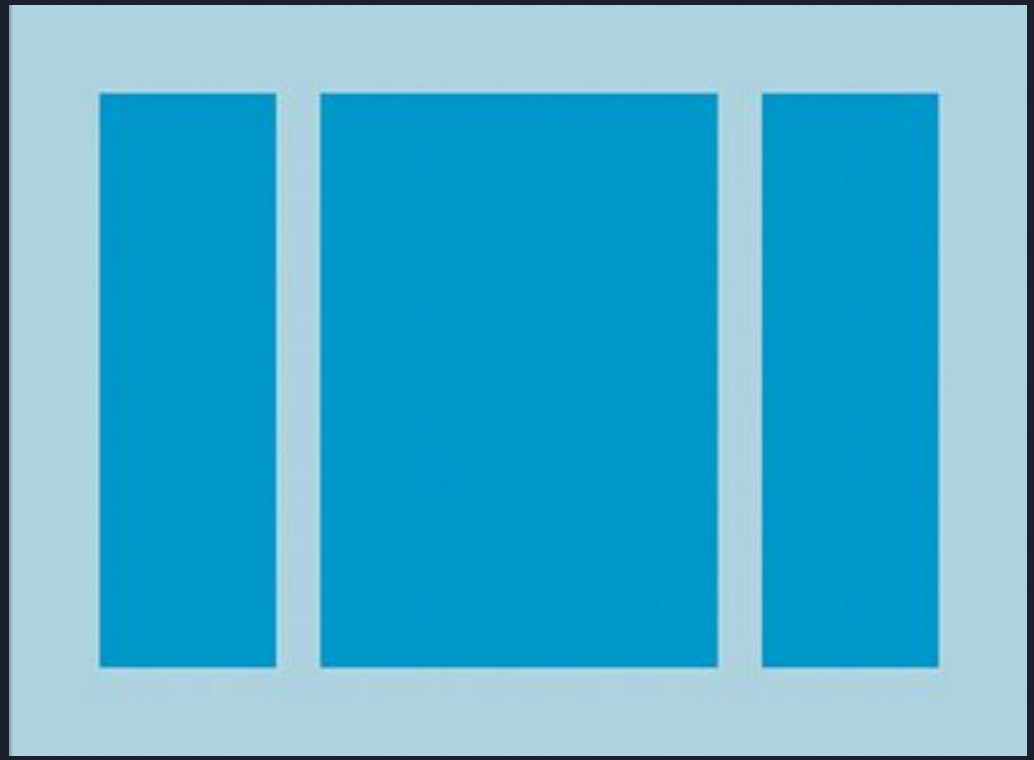


Layout



https://developer.android.com/guide/topics/ui/declaring-layout

Linear Layout



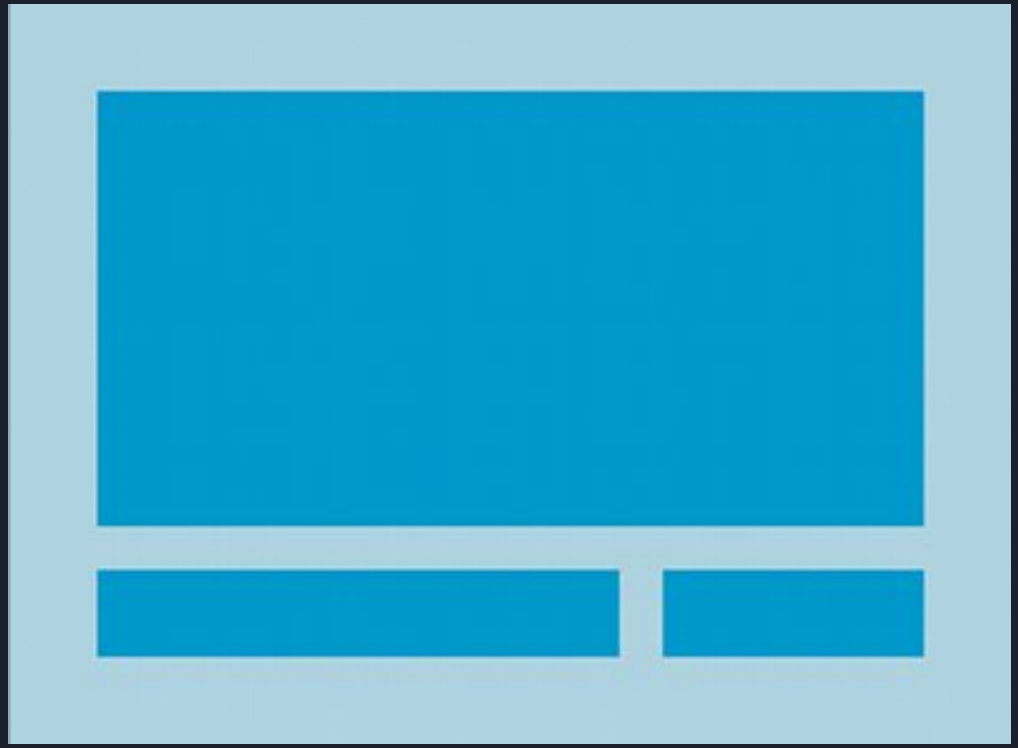
View group that aligns all children in a single direction, vertically or horizontally

https://developer.android.com/guide/topics/ui/layout/linear

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Relative Layout



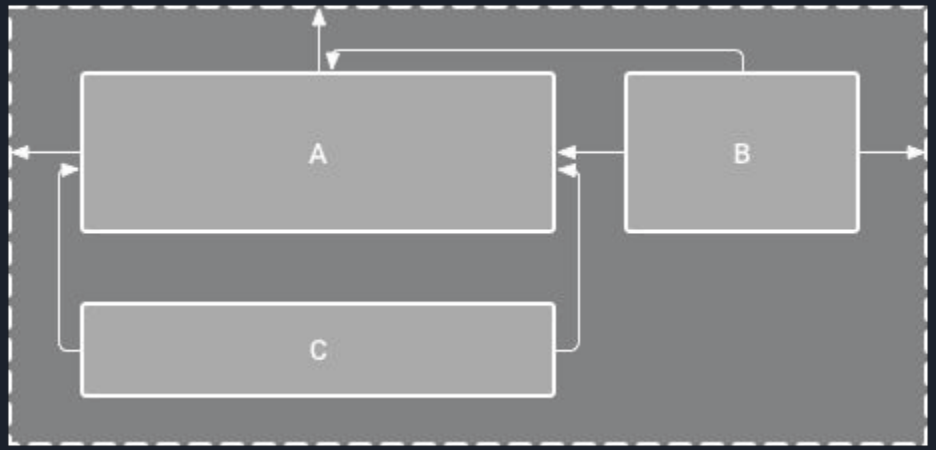
View group that displays child views in relative positions

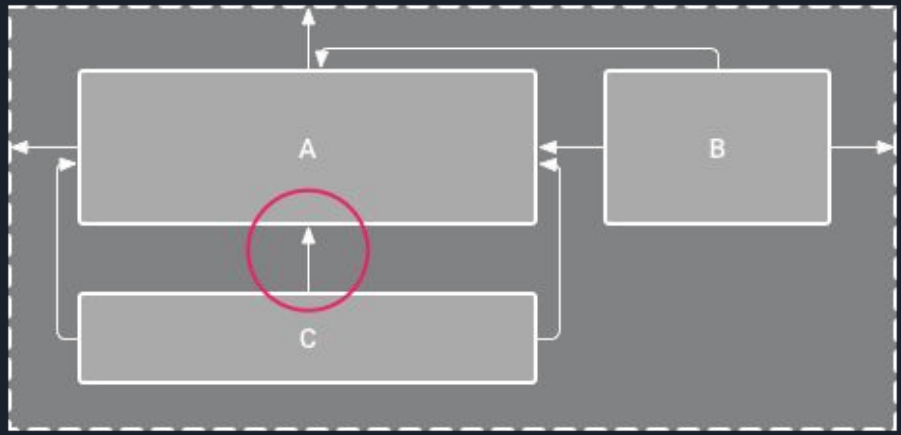
https://developer.android.com/guide/topics/ui/layout/relative

Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

Constraint Layout






```
// build.gradle (app)
dependencies {
    implementation "androidx.constraintlayout:constraintlayout:2.1.0"
}
```

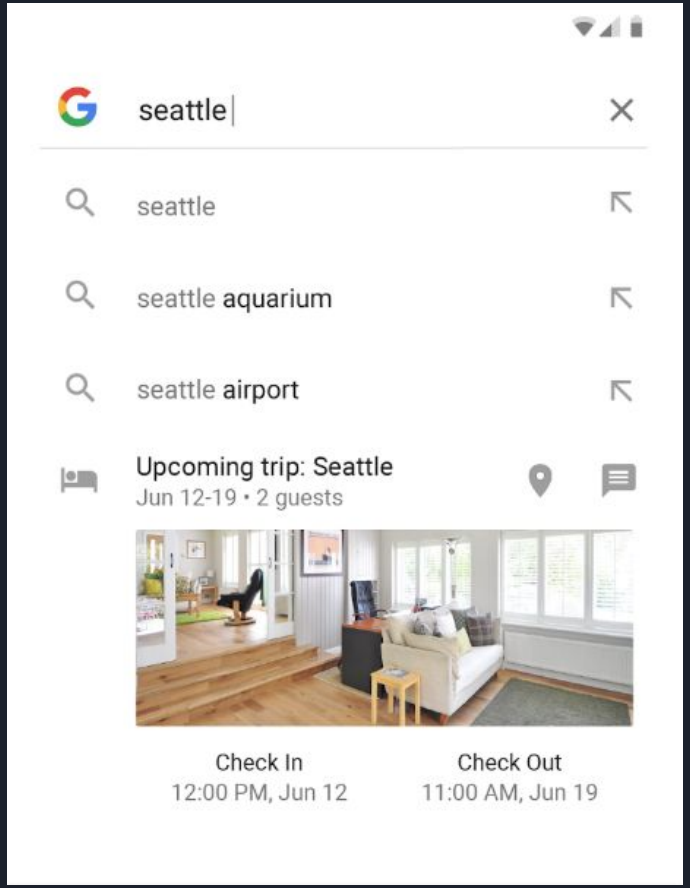
https://developer.android.com/training/constraint-layout

Constraint Layout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

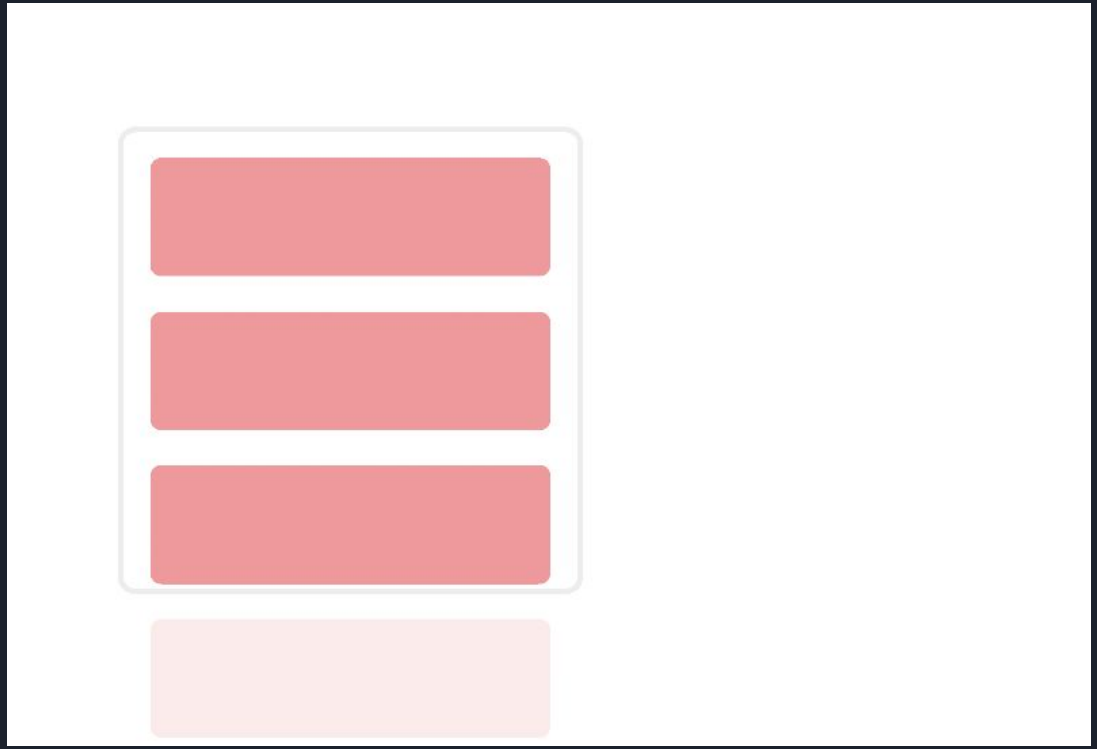
Recycler View

Display large sets of data in your UI while minimizing memory usage.



Recycler View

Display large sets of data in your UI while minimizing memory usage.

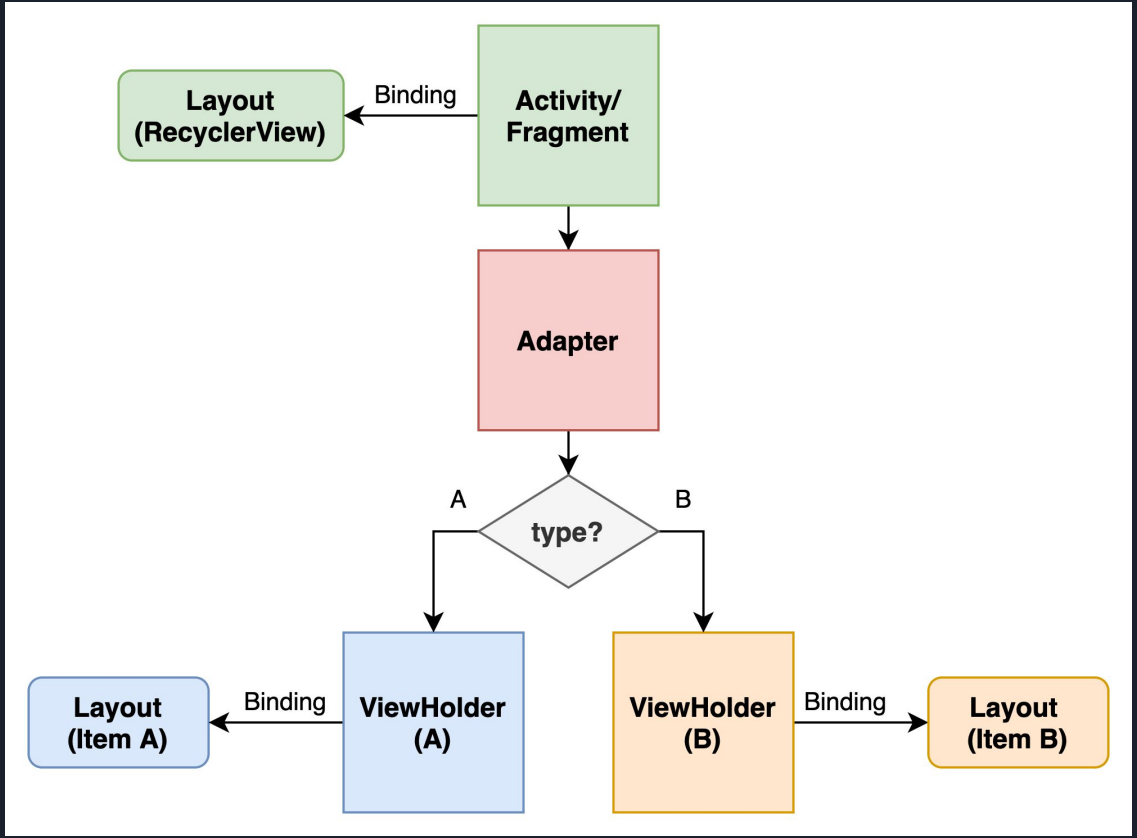


https://akexorcist.dev/content/images/2021/07/recycler_view_in_scroll_view-001.gif

Recycler View

```
// build.gradle (app)
dependencies {
    implementation "androidx.recyclerview:recyclerview:1.2.1"
}
```

Recycler View



View Binding

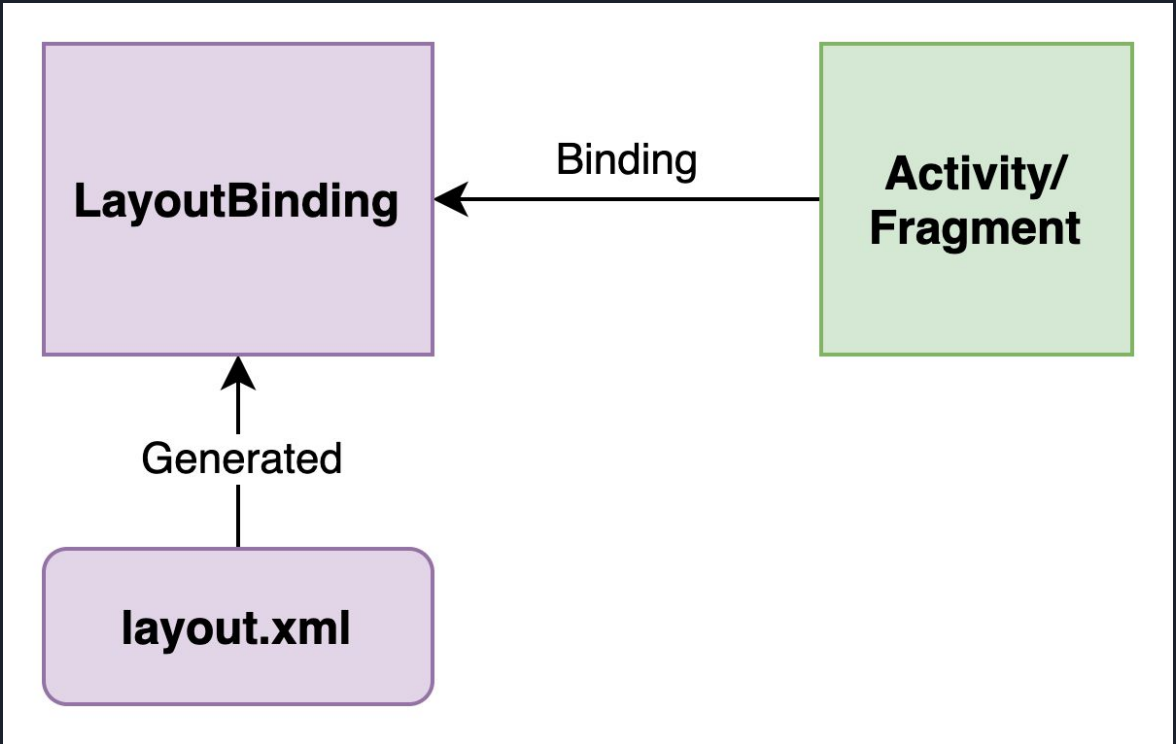
View binding is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a binding class for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

In most cases, view binding replaces **findViewById.**

View Binding

```
// build.gradle (app)

android {
    buildFeatures {
        viewBinding true
    }
}
```

View Binding



Basic Kotlin

```
fun main() {
    println("Hello world!")
}
```

https://kotlinlang.org/docs/basic-syntax.html

Basic Kotlin: Package definition and imports

```
package my.demo

import kotlin.text.*
```

Basic Kotlin: Variables

var/val: Type = value

Read-only local variables are defined using the keyword *val*. They can be assigned a value only once.

```
val a: Int = 1          // immediate assignment
val b = 2               // `Int` type is inferred
val c: Int              // Type required when no initializer is provided
c = 3                   // deferred assignment
```

Variables that can be reassigned use the *var* keyword:

```
var x = 5               // `Int` type is inferred
x += 1
```

```
const val CONST_VALUE = "constant"
```

Basic Kotlin: Comments

```
// This is an end-of-line comment

/* This is a block comment
   on multiple lines. */
```

Basic Kotlin: String templates

```
var a = 1
// simple name in template:
val s1 = "a is $a"

a = 2
// arbitrary expression in template:
val s2 = "${s1.replace("is", "was")}, but now is $a"
println(s2)    // Output: a was 1, but now is 2
```

Basic Kotlin: Control Flow: if

```
// Traditional usage
var max = a
if (a < b) max = b

// With else
var max: Int
if (a > b) {
  max = a
} else {
  max = b
}

// As expression
val max = if (a > b) a else b
```

* no ternary operator (condition ? then : else)

Basic Kotlin: Control Flow: when

```
val x = 1
when (x) {
  1 -> print("x == 1")
  2 -> print("x == 2")
  else -> { // Note the block
    print("x is neither 1 nor 2")
  }
}

val obj: Any = ""
when (obj) {
  1             -> "One"
  "Hello"       -> "Greeting"
  is Long       -> "Long"
  !is String    -> "Not a string"
  else          -> "Unknown"
}
```

Basic Kotlin: Control Flow: for

```
for (item in collection) print(item)

for (item: Int in ints) {
  // ...
}

for (i in 1..3) print(i)
// 123

for (i in 1..8 step 2) print(i)
// 1357

for (i in 6 downTo 0 step 2) {
  print(i)
}
// 6420
```

```
(1..3).forEach {
  print(it)
}
```

* No this format like in java

```
for (int i = first; i <= last; i += step) {
  // ...
}
```

Basic Kotlin: Control Flow: while

```
while (x > 0) {
  x--
}

do {
  val y = 0
} while (y != 0) // y is visible here!
```

Basic Kotlin: Null Safety

```
var a: String = "abc"          // Regular initialization means non-null by default
a = null                       // compilation error

var b: String? = "abc"         // can be set null
b = null                       // ok
```

Safe Calls

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length)             // Unnecessary safe call
```

Basic Kotlin: Null Safety

Elvis Operator (?:)

```
val len: Int = if (b != null) b.length else -1      // Normal checking
val len = b?.length ?: -1                           // Elvis Operator
```

The !! Operator (NPE-lovers)

```
val b: String? = null
val len = b!!.length                                // beware to use
```

Safe Casts

```
val aInt: Int? = a as? Int
```

Basic Kotlin: Functions

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
  .filter { it.startsWith("a") }
  .sortedBy { it }
  .map { it.toUpperCase() }
  .forEach { println(it) }
```

Basic Kotlin: Functions

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
  .filter { it.startsWith("a") }
  .sortedBy { it }
  .map { it.toUpperCase() }
  .forEach { println(it) }

// APPLE
// AVOCADO
```

Basic Kotlin: Functions

```kotlin
// Function having two Int parameters with Int return type:
fun sum1(a: Int, b: Int): Int {
  return a + b
}

// Function with an expression body and inferred return type:
fun sum2(a: Int = 0, b: Int = 1) = a + b

// Function returning no meaningful value:
fun printSum1(a: Int, b: Int): Unit {
  println("sum of $a and $b is ${a + b}")
}

// Unit return type can be omitted:
fun printSum2(a: Int, b: Int) {
  println("sum of $a and $b is ${a + b}")
}
```

Default value

```kotlin
sum1(1, 2)

// call with alias name
sum1(a = 1, b = 2)
```

Basic Kotlin: Functions

```
// Generic functions
fun <T> someList(item: T): List<T> {
  /*...*/
}

// Extension functions
fun String.removeDash(): String {
  return replace("-", "")
}

val str = "12-34".removeDash()
println(str) // 1234
```

Basic Kotlin: Higher-Order Functions

```kotlin
/**
* Performs the given [action] on each element.
*/
public inline fun <T> Iterable<T>.forEach(action: (T) -> Unit): Unit {
  for (element in this) action(element)
}

public inline fun <T> Iterable<T>.forEachIndexed(action: (index: Int, T) -> Unit): Unit {
  var index = 0
  for (item in this) action(checkIndexOverflow(index++), item)
}

val list = listOf(1, 2, 3)
list.forEachIndexed { _, value -> println("$value!") }
```

Function types

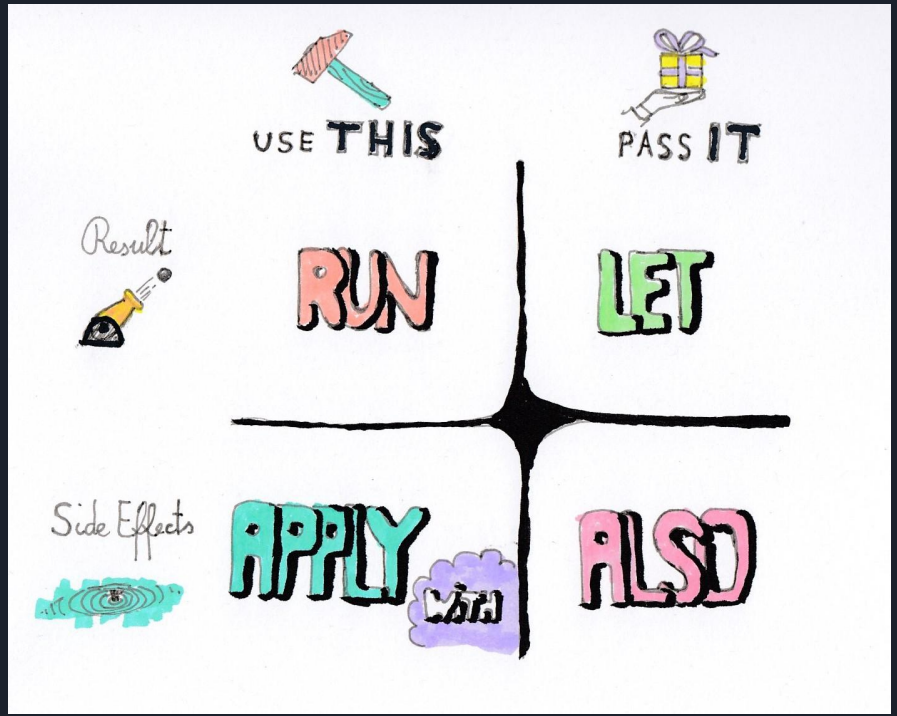
(Int) -> String
() -> Unit

_ parameter is unused

Basic Kotlin: Functions: Lambda expression

```
val sumFull: (Int, Int) -> Int = { x: Int, y: Int -> x + y }
val sum = { x: Int, y: Int -> x + y }

print(sumFull(1, 2))
print(sum(1, 2))
```

Basic Kotlin: Functions: Scope Function



https://medium.com/@ramtop/kotlin-scope-functions-c8c41f09615f

https://kotlinlang.org/docs/scope-functions.html

Basic Kotlin: Classes

```
interface Shapable {
  fun area(): Double
}

open class Shape : Shapable {
  open val vertexCount: Int = 0

  open fun draw() {
    println("draw on Shape $vertexCount")
  }

  fun fill() {
    println("fill on Shape")
  }

  override fun area(): Double {
    return 0.0
  }
}
```

private means visible inside this class only (including all its members);

protected — same as private + visible in subclasses too;

internal — any client inside this module who sees the declaring class sees its internal members;

public — any client who sees the declaring class sees its public members.

```
val shape = Shape()
shape.draw()
```

Basic Kotlin: Classes

```
class Rectangle : Shape() {

  override val vertexCount = 4

  override fun draw() {
    println("call super.draw() on Rectangle")
    super.draw()
  }
}
```

Basic Kotlin: Classes

```
interface Circleable {
  var radius: Double
}

class Circle : Shape(), Circleable {

  override var radius: Double = 2.0

  override fun draw() {
    println("draw on Circle $vertexCount")
  }

  override fun area(): Double {
    return PI * radius * radius
  }
}
```

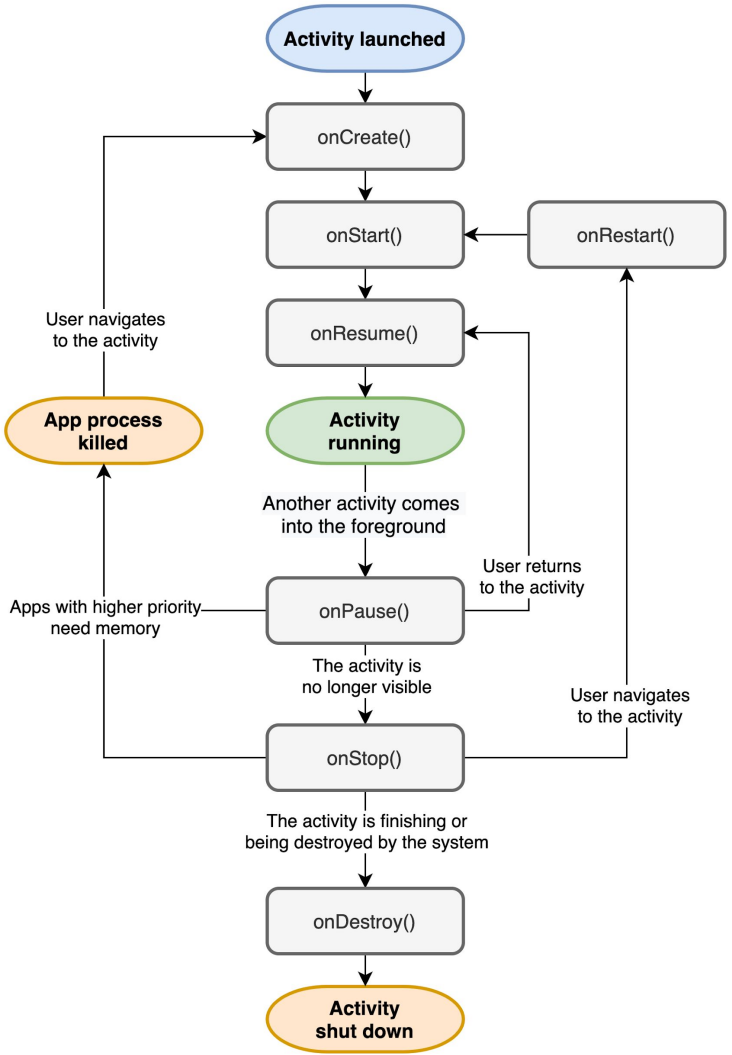
Basic Kotlin: Classes

```
class ClassStatic {

  val normalValue = "this is normal value"

  companion object {
    const val STATIC_VALUE = "this is static value"
  }
}

object ClassObject {
  const val STATIC_VALUE = "this is static value"
}

data class User(val name: String, val age: Int)
```

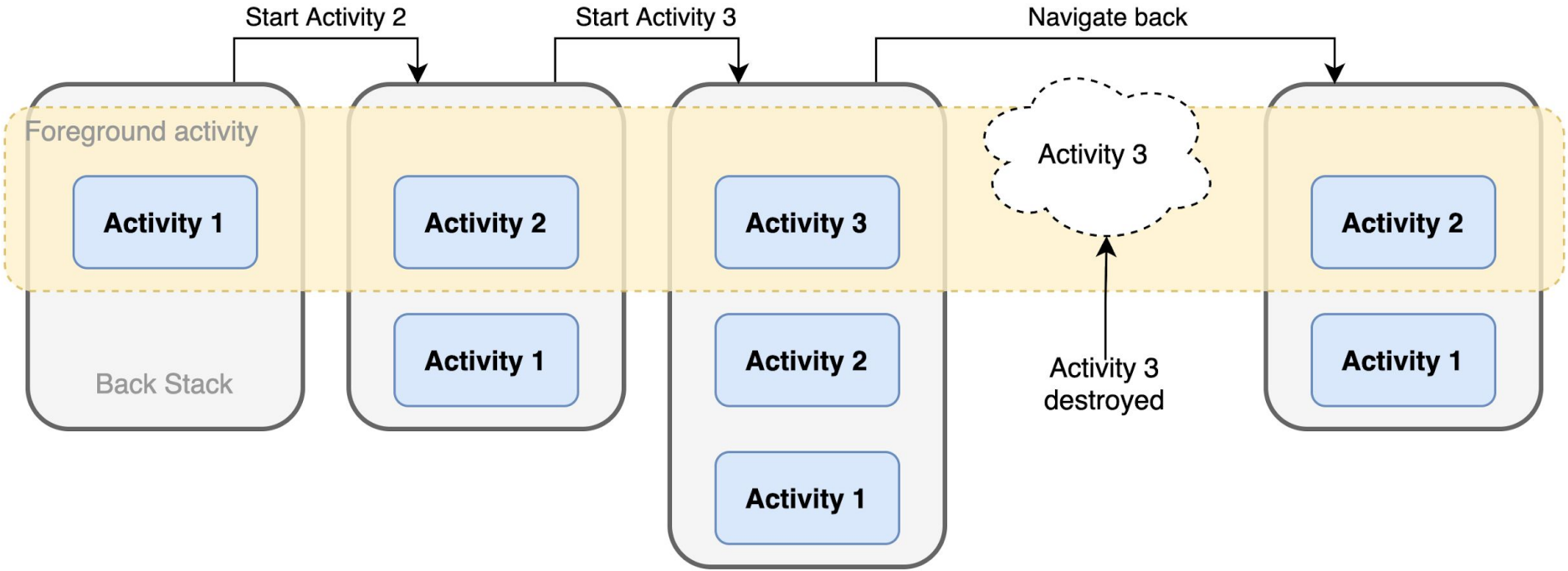
Activity

The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model.

Activity: Lifecycle



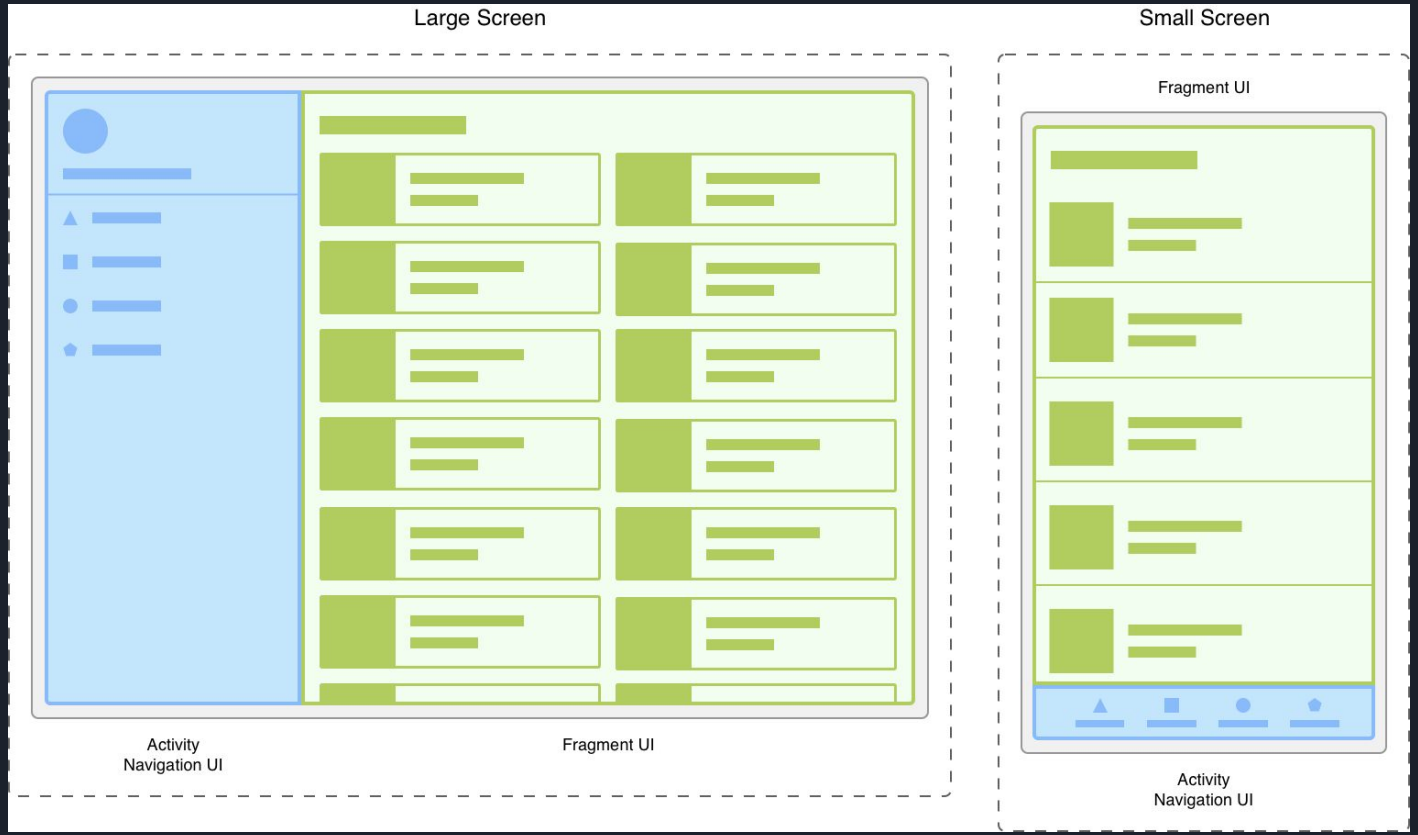
Activity: Navigation



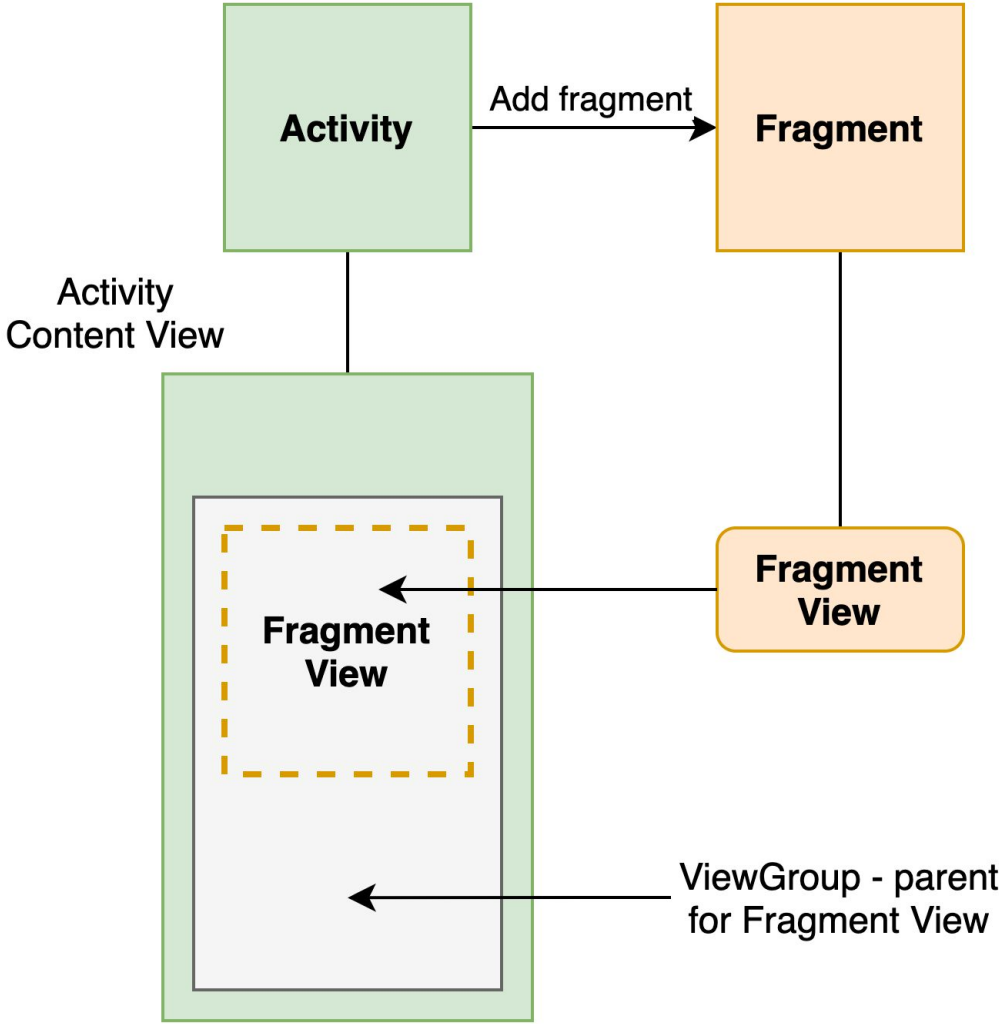
Fragment

A **Fragment** represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be hosted by an activity or another fragment. The fragment’s view hierarchy becomes part of, or attaches to, the host’s view hierarchy.

Fragment


Large Screen
Small Screen
Fragment UI
Activity
Navigation UI
Fragment UI
Activity
Navigation UI

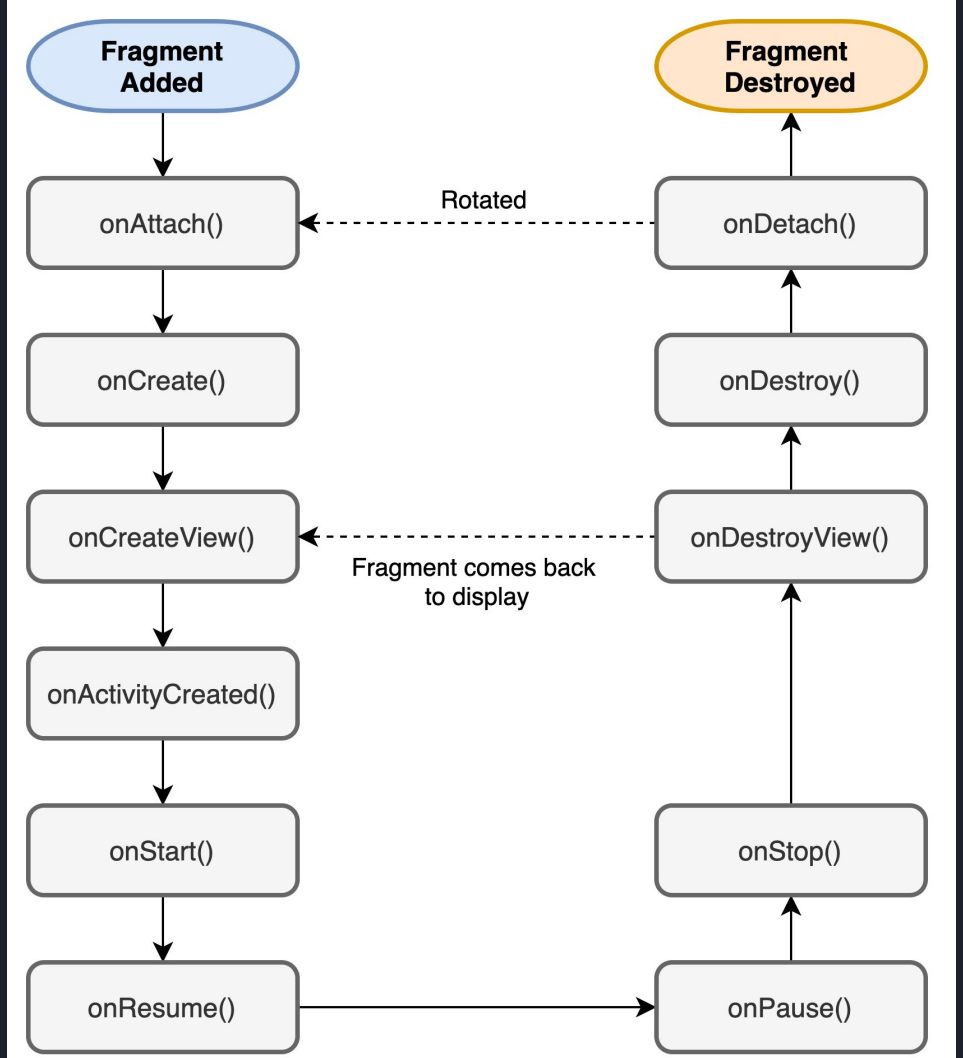
Fragment


Activity
Add fragment
Fragment
Activity
Content View
Fragment
View
Fragment
View
ViewGroup - parent
for Fragment View

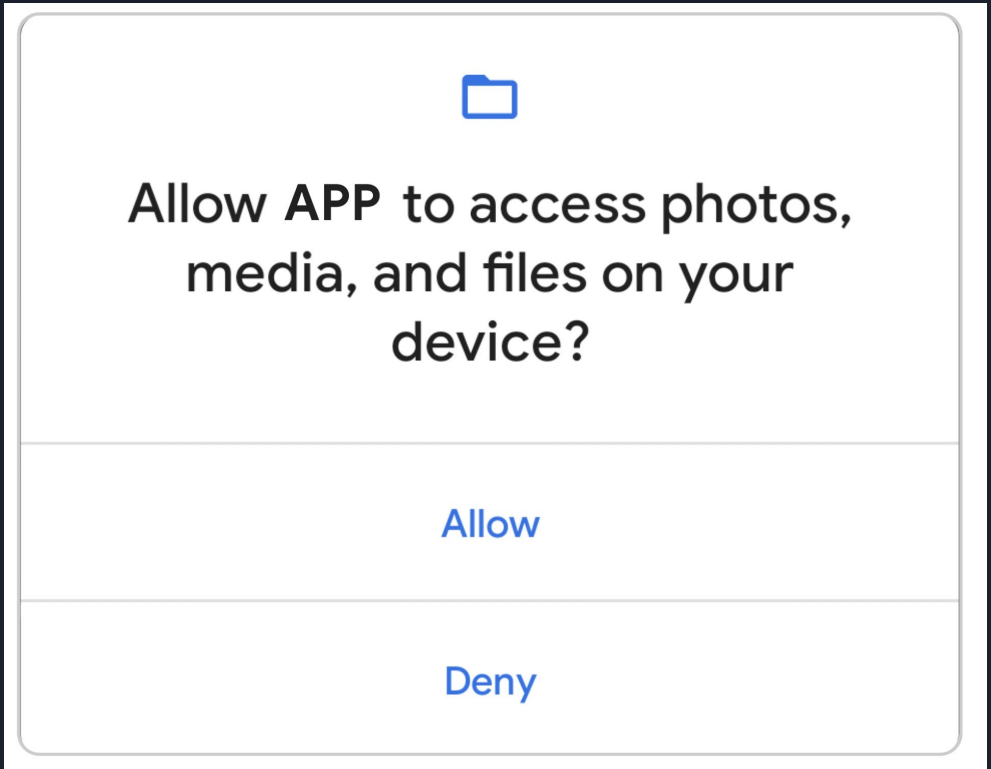
Fragment

```
// build.gradle (app)
dependencies {
    def fragment_version = "1.3.6"
    implementation "androidx.fragment:fragment-ktx:$fragment_version"
}
```

Fragment: Lifecycle

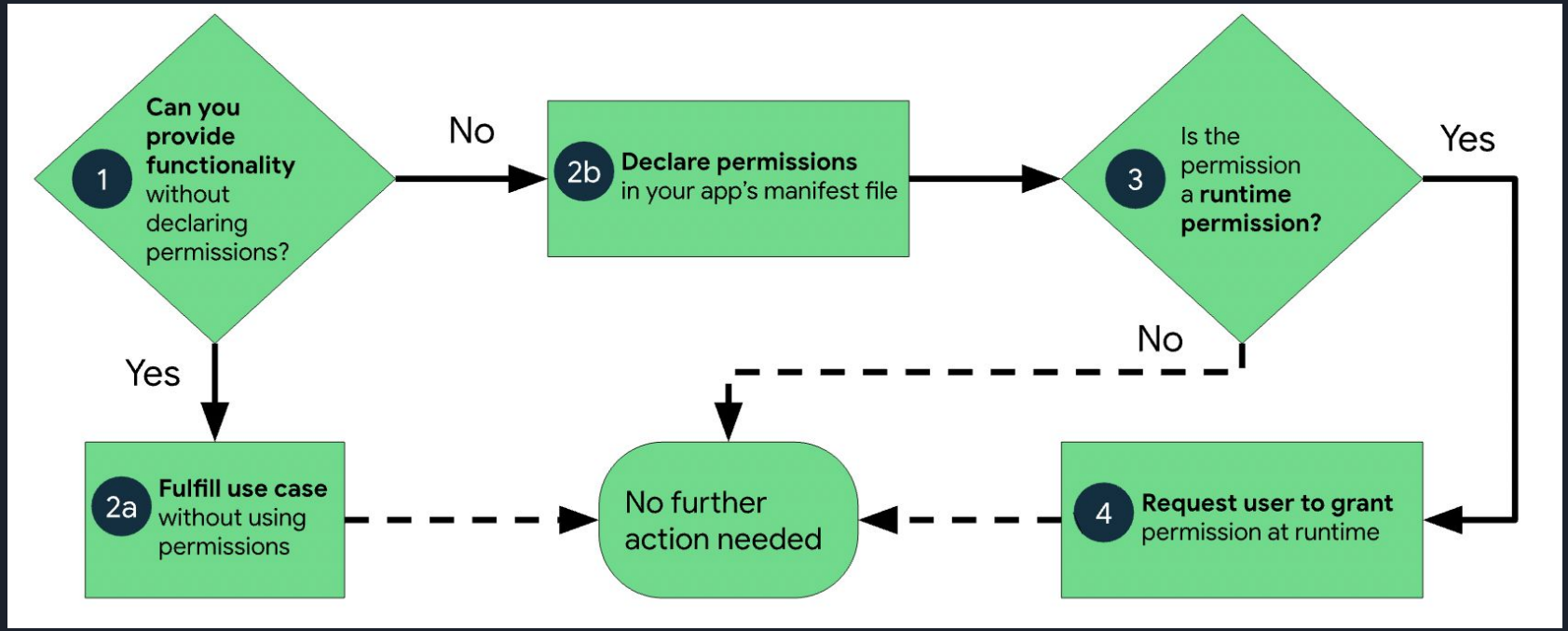


Permissions



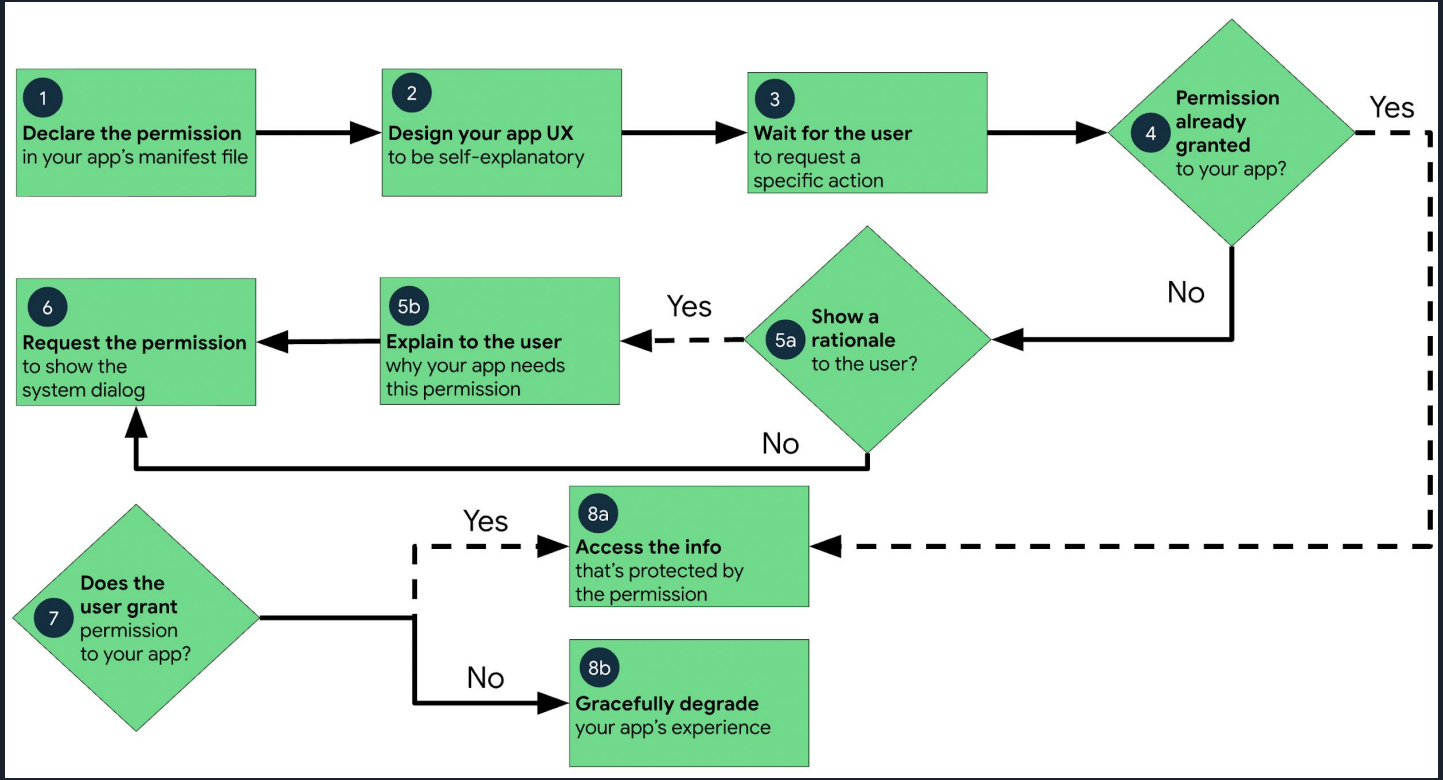
https://developer.android.com/guide/topics/permissions/overview

Permissions



https://developer.android.com/guide/topics/permissions/overview

Permissions



https://developer.android.com/training/permissions/requesting

Permissions

```
// AndroidManifest.xml

<manifest ...>
<!-- Add declaration to app manifest -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />

<!--  Declare hardware as optional -->
<uses-feature android:name="android.hardware.camera"
                    android:required="false" />

  <application ...>
    ...
  </application>
</manifest>
```

https://developer.android.com/training/permissions/requesting

Permissions

Dexter
https://github.com/Karumi/Dexter

PermissionsDispatcher
https://github.com/permissions-dispatcher/PermissionsDispatcher

RxPermissions
https://github.com/tbruyelle/RxPermissions

https://developer.android.com/training/permissions/requesting

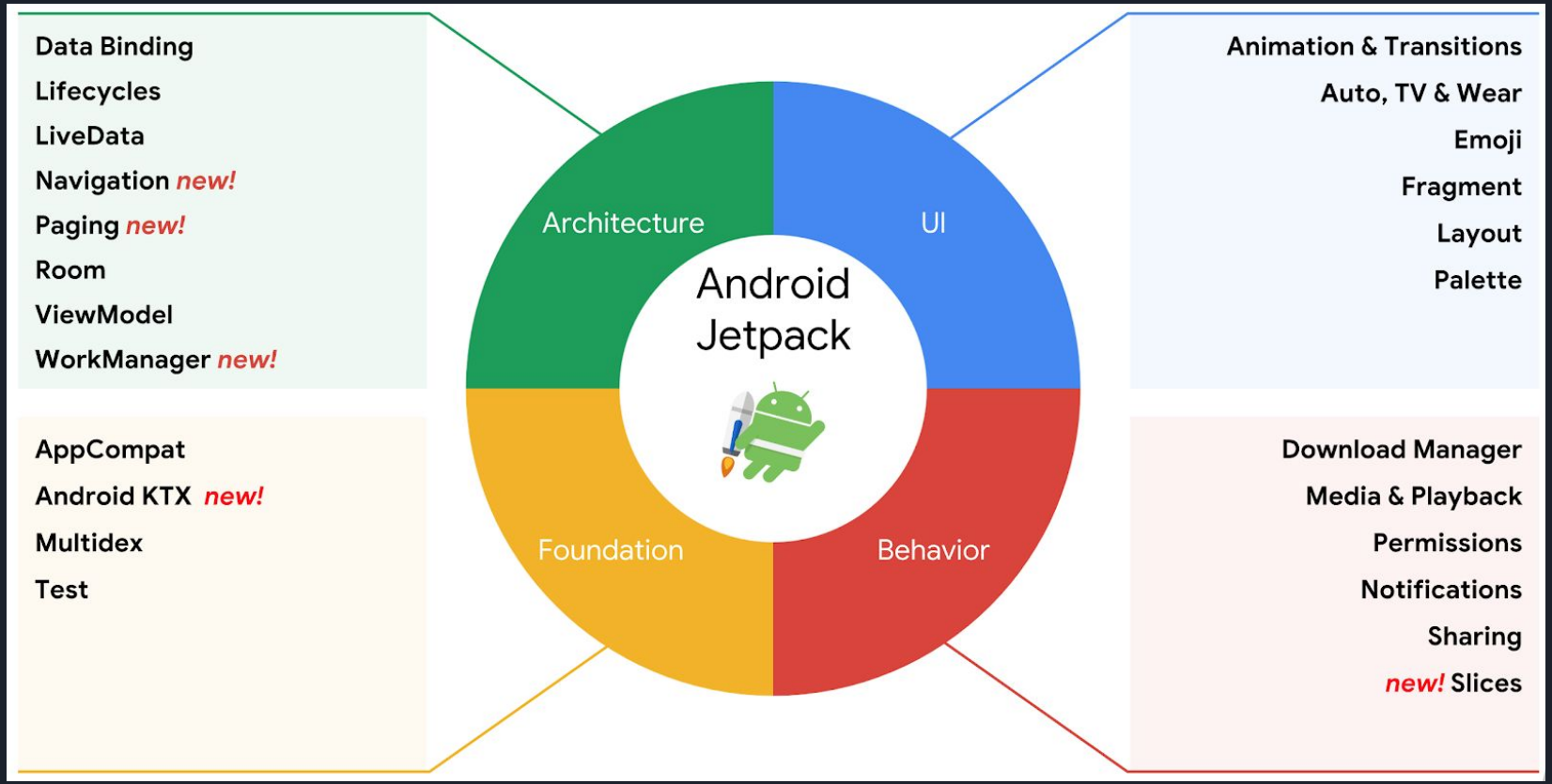
Android Jetpack

Android Jetpack

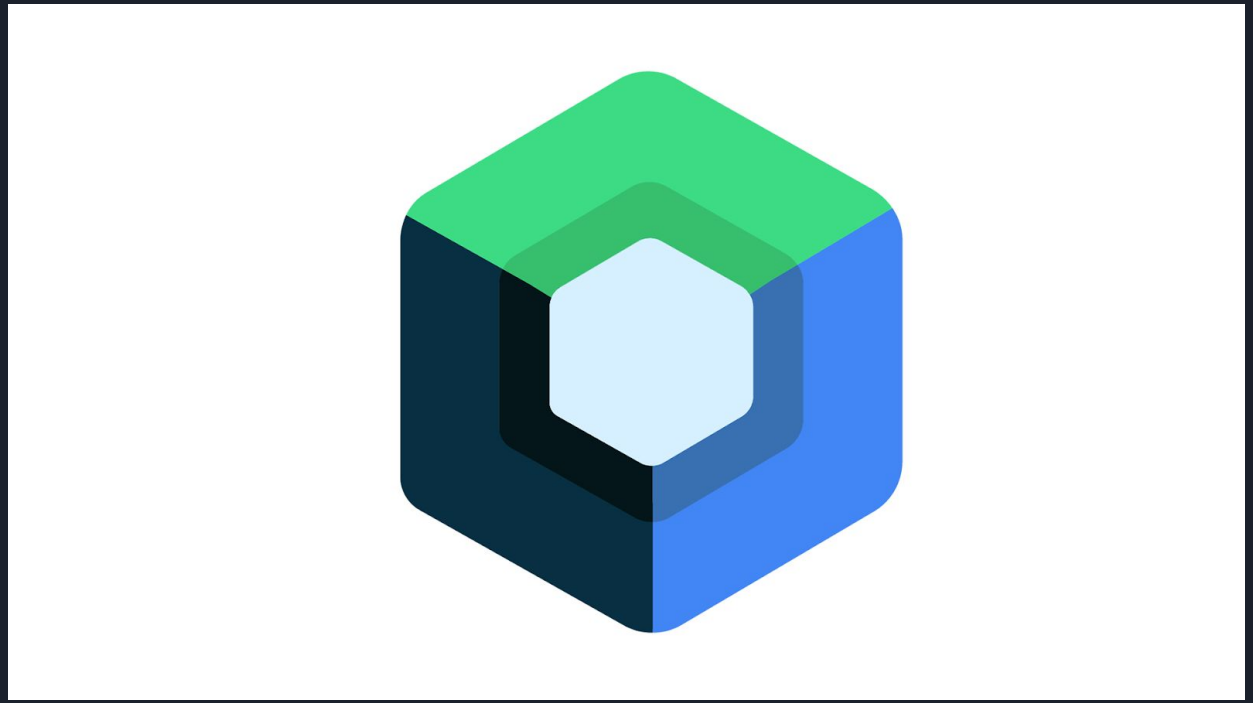
Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.

https://developer.android.com/jetpack

Android Jetpack



Jetpack Compose

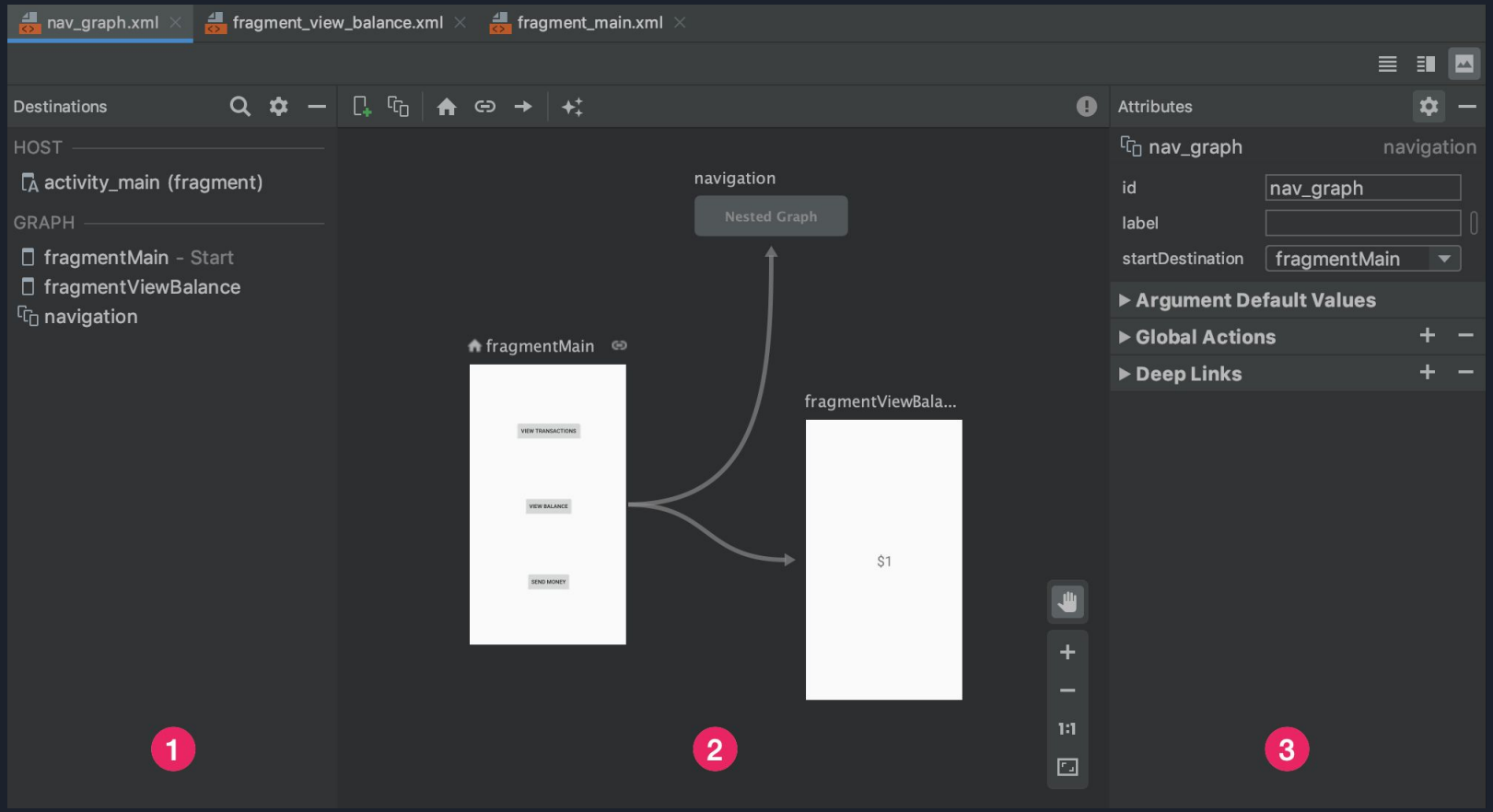


https://developer.android.com/jetpack/compose

Jetpack Compose

Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.

Navigation Component



Navigation Component

The **Navigation component** consists of three key parts that are described below:

- **Navigation graph**: An XML resource 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.
- **NavHost**: An empty container that displays destinations from your navigation graph. The Navigation component contains a default NavHost implementation, **NavHostFragment**, that displays fragment destinations.
- **NavController**: An object that manages app navigation within a NavHost. The NavController orchestrates the swapping of destination content in the NavHost as users move throughout your app.

https://developer.android.com/guide/navigation

Data Binding

The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.

https://developer.android.com/topic/libraries/data-binding

Data Binding

```
// build.gradle (app)

android {
    buildFeatures {
        dataBinding true
    }
}
```

Data Binding

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
  <data>
    <variable
      name="viewmodel"
      type="com.myapp.data.ViewModel" />
  </data>

  <ConstraintLayout... >
     <TextView
          android:text="@{viewmodel.userName}" />
  </ConstraintLayout>
</layout>
```

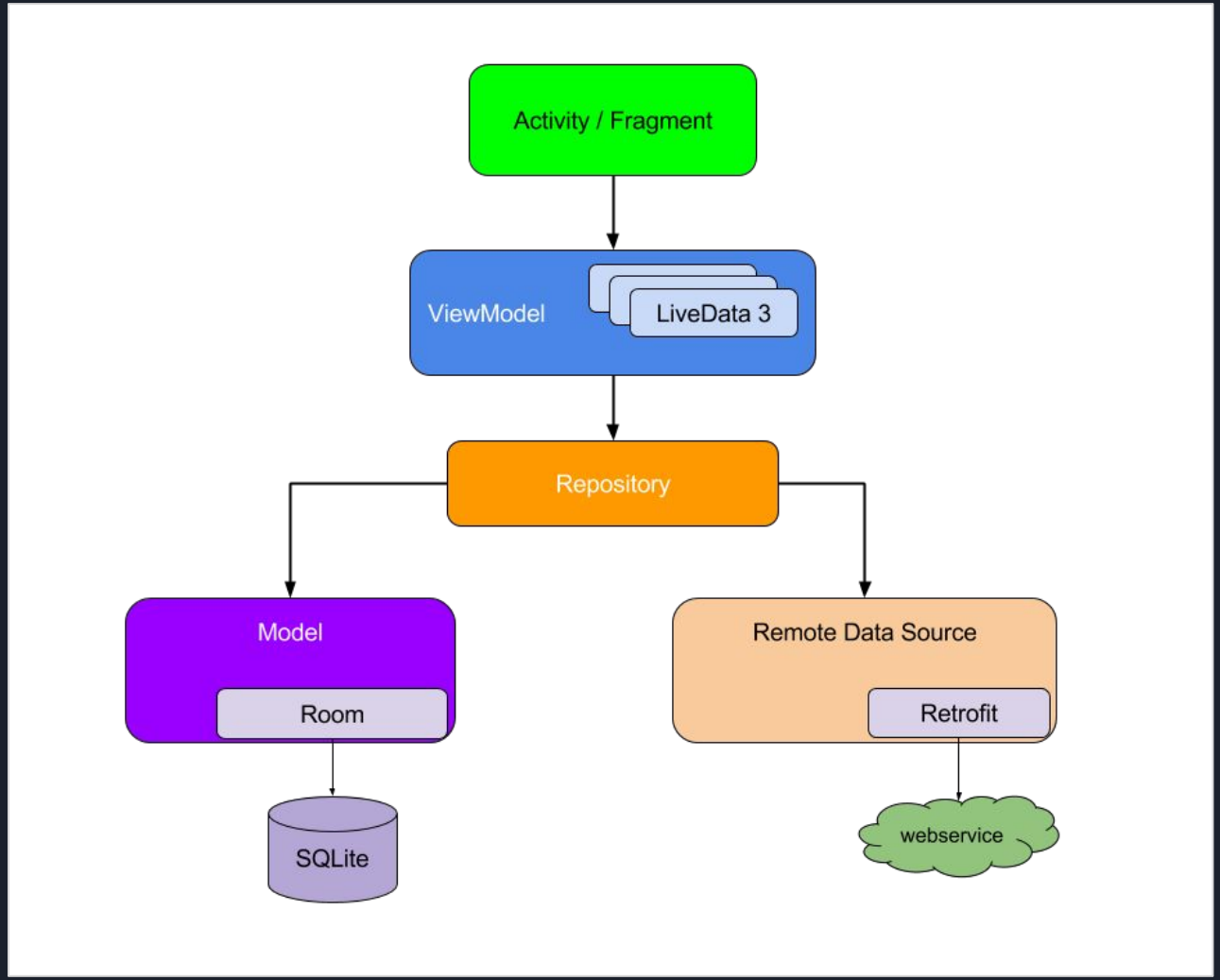
Data Binding

Two-way data binding:
https://developer.android.com/topic/libraries/data-binding/two-way

Android Architecture

Android architecture components are a collection of libraries that help you design robust, testable, and maintainable apps. Start with classes for managing your UI component lifecycle and handling data persistence.

https://developer.android.com/topic/libraries/architecture


Activity / Fragment
ViewModel
LiveData 3
Repository
Model
Room
SQLite
Remote Data Source
Retrofit
webservice

LiveData

LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

https://developer.android.com/topic/libraries/architecture/viewmodel

LiveData

```
// build.gradle (app)
dependencies {
        implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.3.1"
}
```

https://developer.android.com/topic/libraries/architecture/viewmodel

ViewModel

The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way.

The ViewModel class allows data to survive configuration changes such as screen rotations.

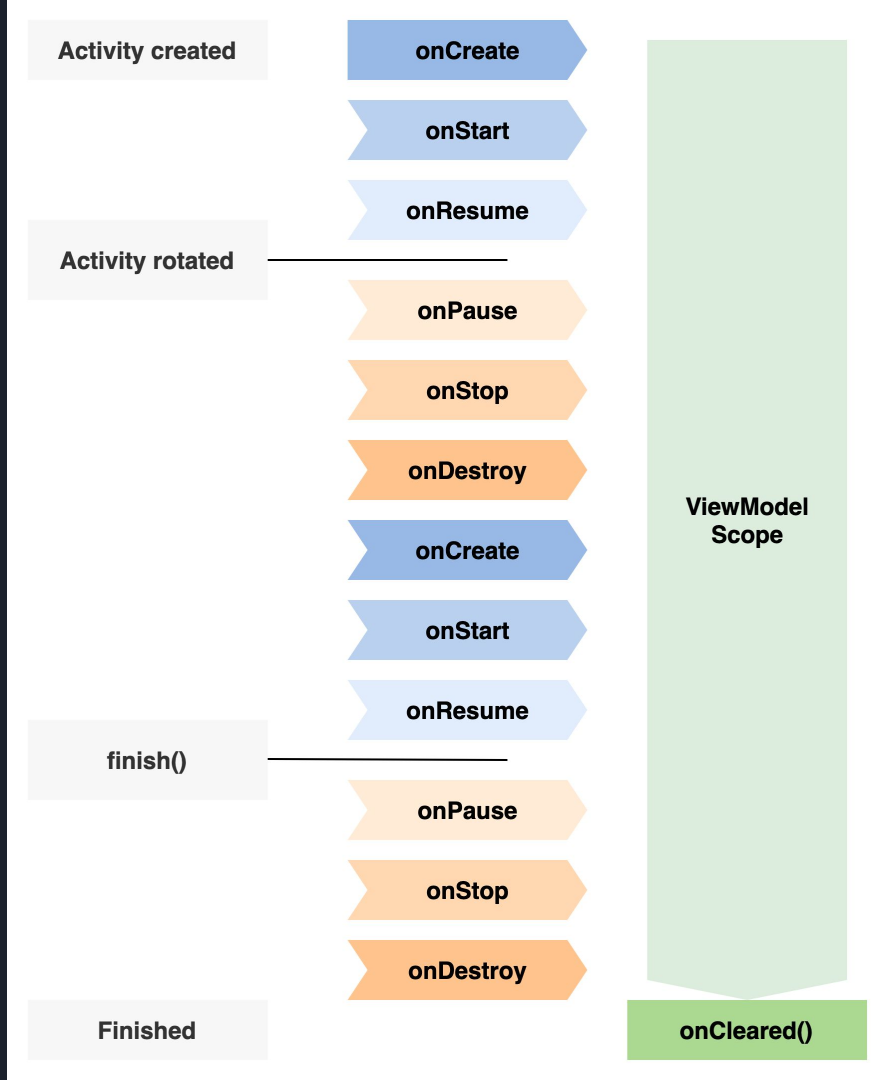
https://developer.android.com/topic/libraries/architecture/viewmodel

ViewModel

```
// build.gradle (app)
dependencies {
        def lifecycle_version = '2.3.1'
        implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1"
}
```

https://developer.android.com/topic/libraries/architecture/viewmodel

ViewModel


Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

Network

- **OkHttp**: https://square.github.io/okhttp/

  OkHttp is an HTTP client that's efficient by default

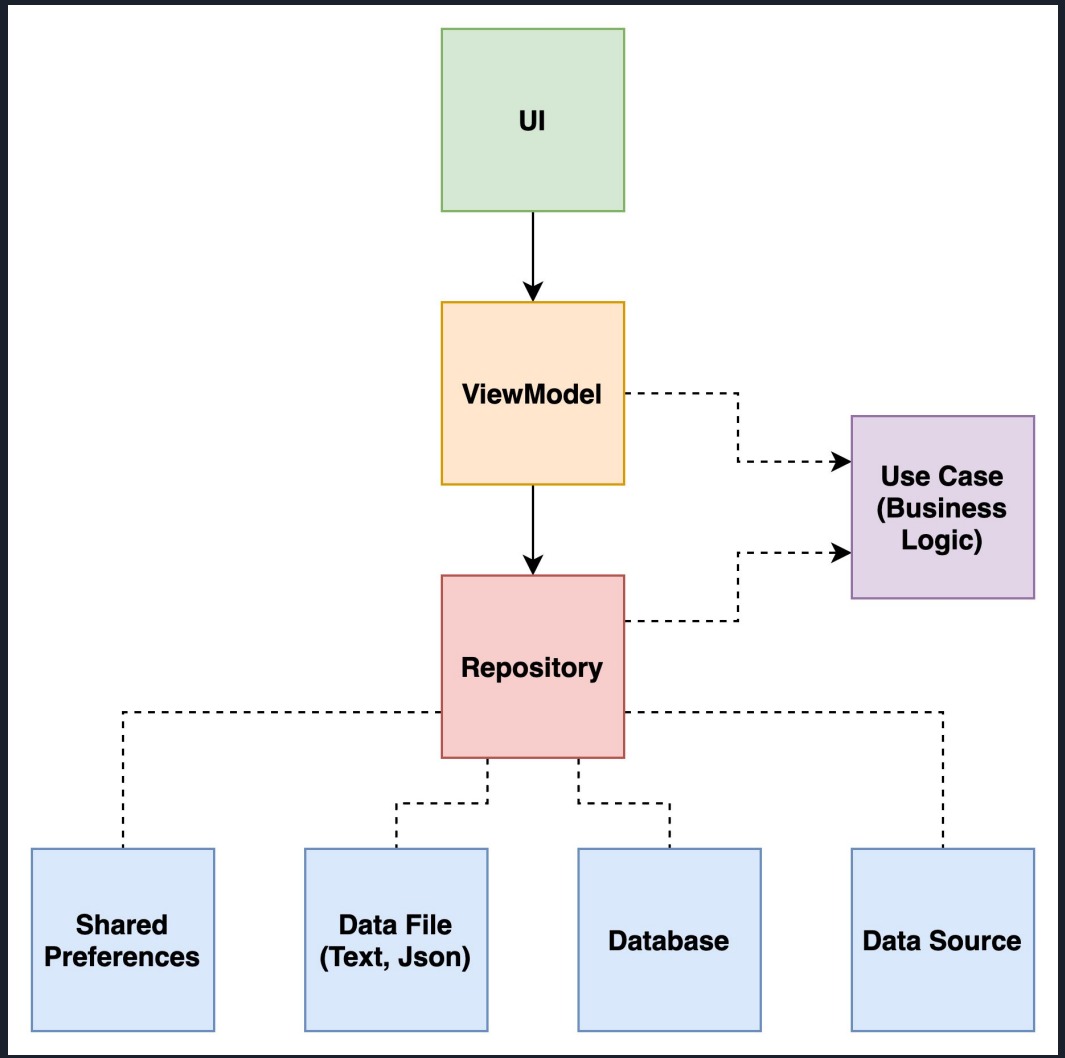
- **Retrofit**: https://square.github.io/retrofit/

  A type-safe HTTP client for Android and Java

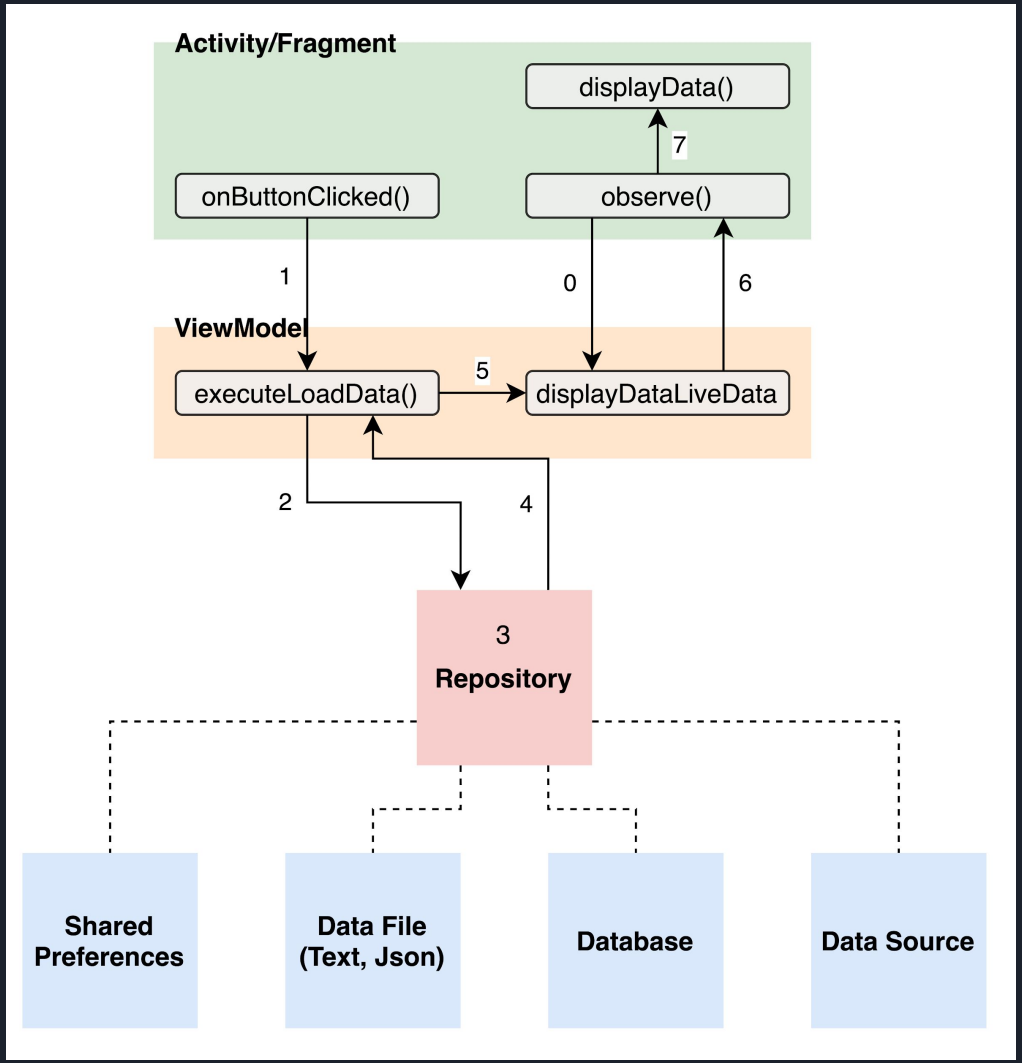
- **Gson**: https://github.com/google/gson

  Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

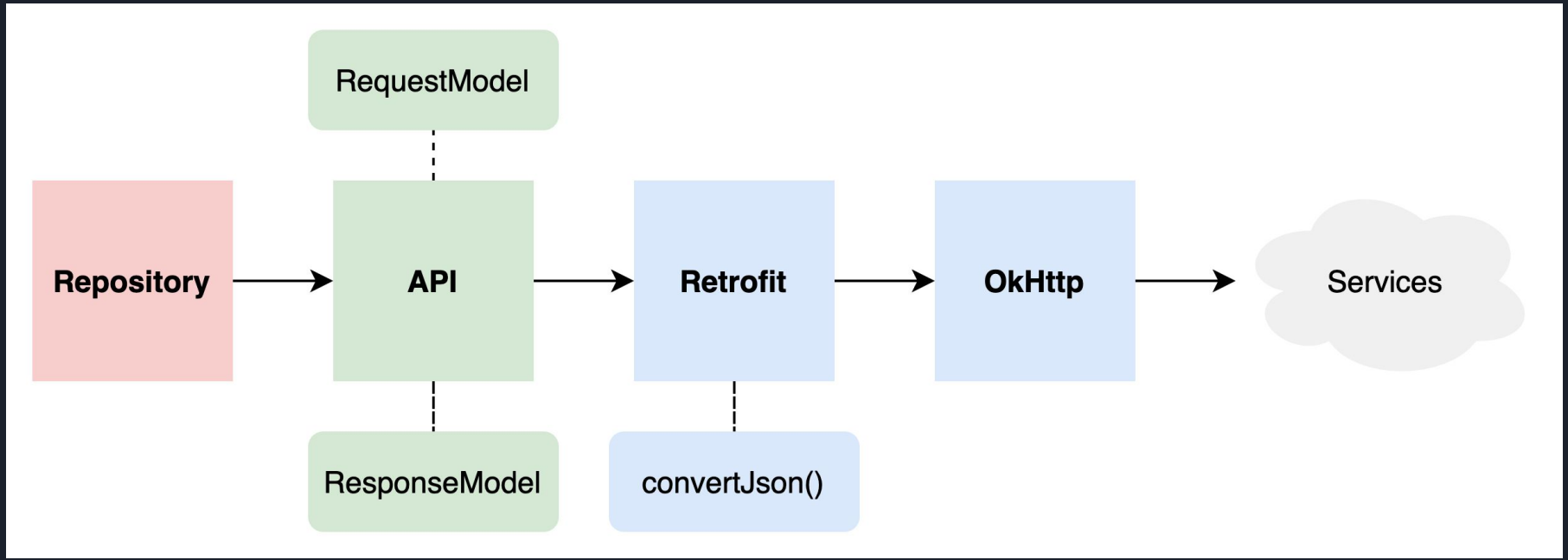
Structure workflow



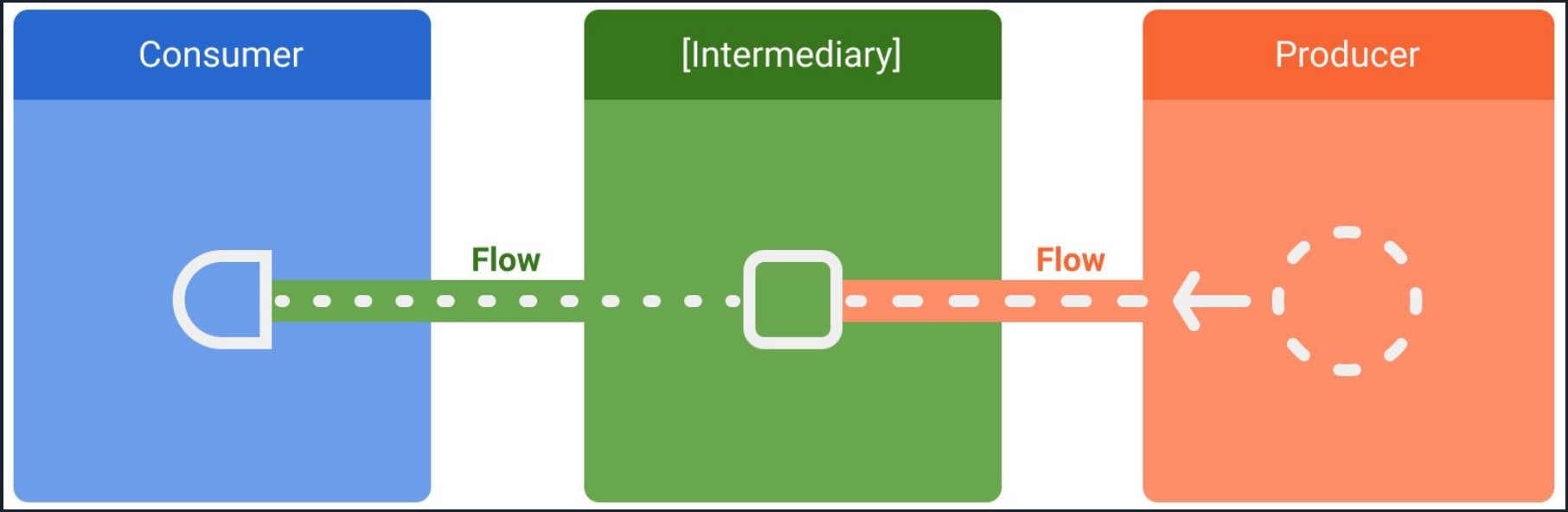
Structure workflow



Structure workflow



Kotlin: Coroutines Flows



Kotlin: Coroutines Flows

A Flow is an async sequence of values

Flow produces values one at a time (instead of all at once) that can generate values from async operations like network requests, database calls, or other async code. It supports coroutines throughout its API, so you can transform a flow using coroutines as well!

Flow can be used in a fully-reactive programming style. If you've used something like RxJava before, Flow provides similar functionality. Application logic can be expressed succinctly by transforming a flow with functional operators such as **map**, **flatMapLatest**, **combine**, and so on.

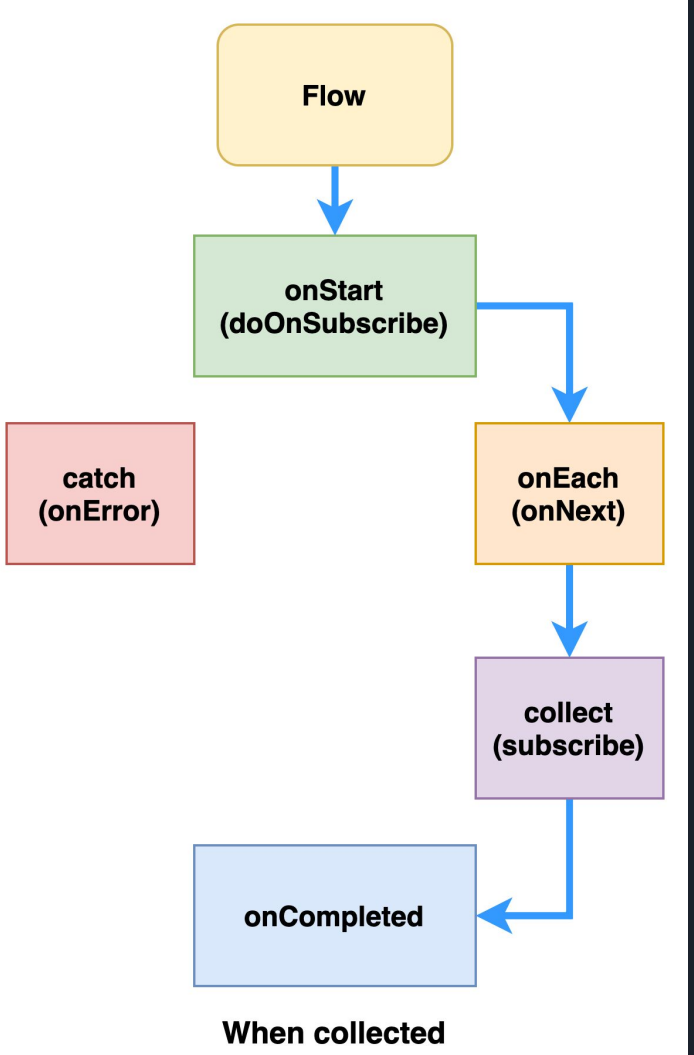
https://codelabs.developers.google.com/codelabs/advanced-kotlin-coroutines#7

Kotlin: Coroutines Flows

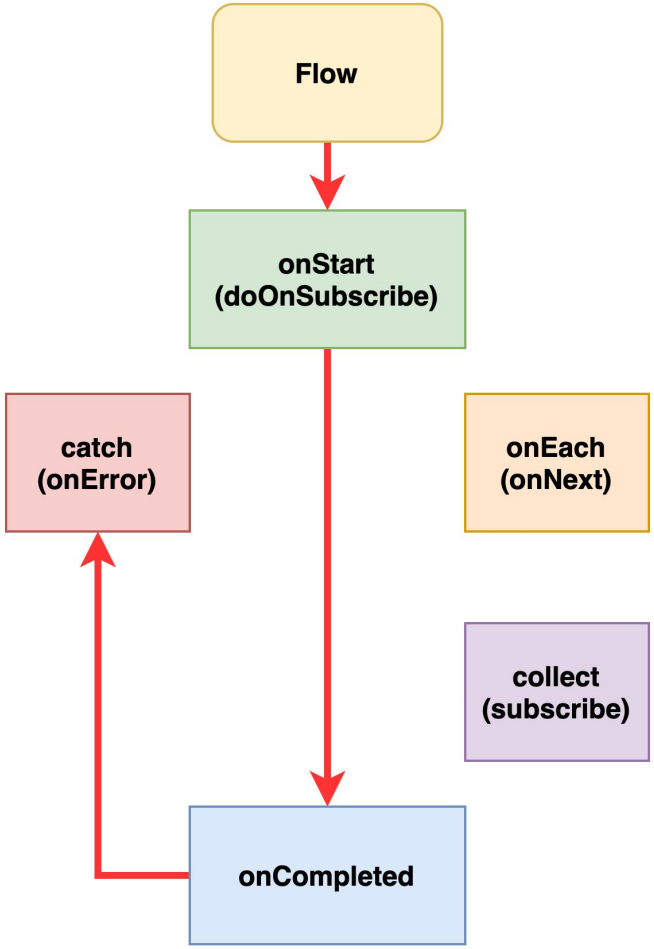
```
// build.gradle (app)
dependencies {
        implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.1"
        implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.4.1"
}
```

https://codelabs.developers.google.com/codelabs/advanced-kotlin-coroutines#7

Kotlin:
Coroutines Flows



Kotlin:
Coroutines Flows



Dependency Injection

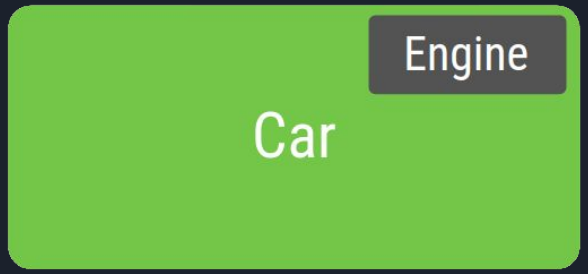
Dependency injection (DI) is a technique widely used in programming and well suited to Android development. By following the principles of DI, you lay the groundwork for good app architecture.

Implementing dependency injection provides you with the following advantages:

- Reusability of code
- Ease of refactoring
- Ease of testing

https://developer.android.com/training/dependency-injection

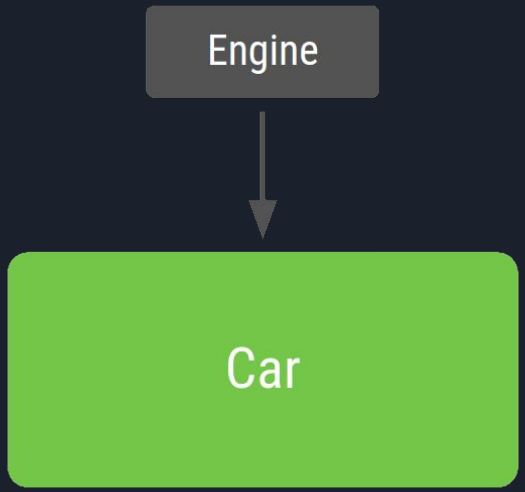
Dependency Injection: Without




```
class Car {

  private val engine = Engine()

  fun start() {
    engine.start()
  }
}

fun main(args: Array) {
  val car = Car()
  car.start()
}
```

https://developer.android.com/training/dependency-injection

Dependency Injection



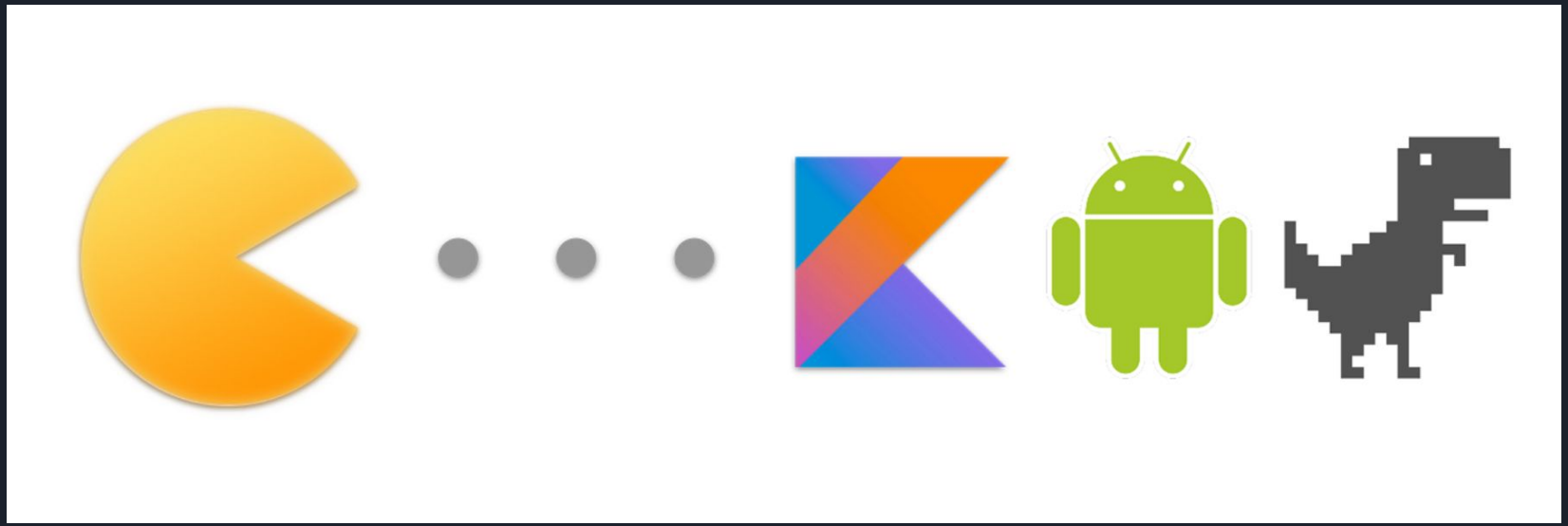

```
class Car(private val engine: Engine) {
    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val engine = Engine()
    val car = Car(engine)
    car.start()
}
```

https://developer.android.com/training/dependency-injection

Dependency Injection: Hilt

https://developer.android.com/training/dependency-injection/hilt-android

Dependency Injection: Koin



https://insert-koin.io/

Mock Response

Stubby4j