

# **Desarrollo de Aplicaciones con Android**

Cristian Mamani

2021



## **Horarios**

Todos los Jueves de 19.00 a 21.00

Todos los Sábados de 10.00 a 12.00

Desde el 26 de Agosto hasta el 16 de Diciembre

# Temario

0. Introduction to Android
1. Build your first App
2. Layouts
3. Navigation
4. Activity and fragment life cycles
5. Architecture
6. Room database and coroutines
7. RecyclerView
8. Connecting to the Internet
9. Repository
10. Design

# Evaluación

Parámetros	Puntaje
Codelabs	5
Primera Prueba (Sept)	10
Codelabs	7
Segunda Prueba (Oct)	10
Codelabs	8
Tercera Prueba (Nov)	10
Trabajo Final (Dic)	50
<b>TOTAL</b>	100

**Nota de aprobación: 80**

# Trabajo Final

→ Realizar una aplicación de cualquier tipo que tenga por lo menos 4 de las siguientes características:

- ◆ Activities
- ◆ Fragments
- ◆ App Architecture (ViewModel, LiveData, Databinding)
- ◆ Room
- ◆ Recycler View
- ◆ Retrofit
- ◆ Notifications
- ◆ GPS
- ◆ Services
- ◆ Content Providers
- ◆ BroadCast Receiver

# **Diagnóstico**

<https://forms.gle/GHb3wJPV3HHBMJh>

Q7

# **Codelab # 1: Install Android Studio**

<https://bit.ly/2HYbe83>

# Android

<https://www.android.com/>

# Introducción a Android

- Sistema Operativo
- Google
- Dispositivos móviles
- Kernel de Linux
- Open Source

# **Historia de Android**

- 2011: Android 4.0 - Ice Cream Sandwich
- 2012: Android 4.1 - Jelly Bean
- 2013: Android 4.4 - Kit Kat
- 2014: Android 5.0 - Lollipop
- 2015: Android 6.0 - Marshmallow
- 2016: Android 7.0 - Nougat
- 2017: Android 8.0 - Oreo
- 2018: Android 9.0 - Pie
- 2019: Android 10.0 - Q
- 2020: Android 11

# Retos de desarrollar Apps

## → Disponibilidad

- ◆ Fragmentación
- ◆ Versiones de Sistemas Operativos
- ◆ Densidad

## → Rendimiento

- ◆ Memoria
- ◆ CPU

## → Calidad

- ◆ Que la aplicación haga lo que dice que hace
- ◆ Workflow
- ◆ Métricas de Diseño

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.8%
4.2 Jelly Bean	17	99.2%
4.3 Jelly Bean	18	98.4%
4.4 KitKat	19	98.1%
5.0 Lollipop	21	94.1%
5.1 Lollipop	22	92.3%
6.0 Marshmallow	23	84.9%
7.0 Nougat	24	73.7%
7.1 Nougat	25	66.2%
8.0 Oreo	26	60.8%
8.1 Oreo	27	53.5%
9.0 Pie	28	39.5%
10. Android 10	29	8.2%

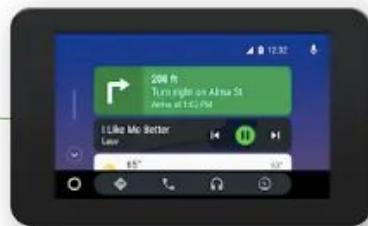
# Tipos de dispositivos Android y Soporte



Phones & Tablets →



WearOS by Google ↗



Android Auto →



Android TV →

# Compatibilidad

→ Con el dispositivo.

- ◆ Características de Hardware
- ◆ Configuración de Pantalla

→ Con el sistema.

- ◆ Version de Android
  - minSDKVersion. Ej. 15
  - targetSDKVersion. Ej. 26

# Usabilidad

## Mobile Usability



49% One Handed



36% Cradled



15% Two Handed

Hard to reach  
one handed or  
with thumb

HARD

OK

EASY

Comfortable  
to reach

@designxp.co

## **Tipos de Almacenamiento**

- Interno
- Externo

# Niveles de Almacenamiento

- File
- SharedPreferences
- Bases de Datos
- Content Providers
- Web services

# Lenguajes de Programación

- Kotlin
- Java
- C++

# Kotlin

<https://kotlinlang.org>

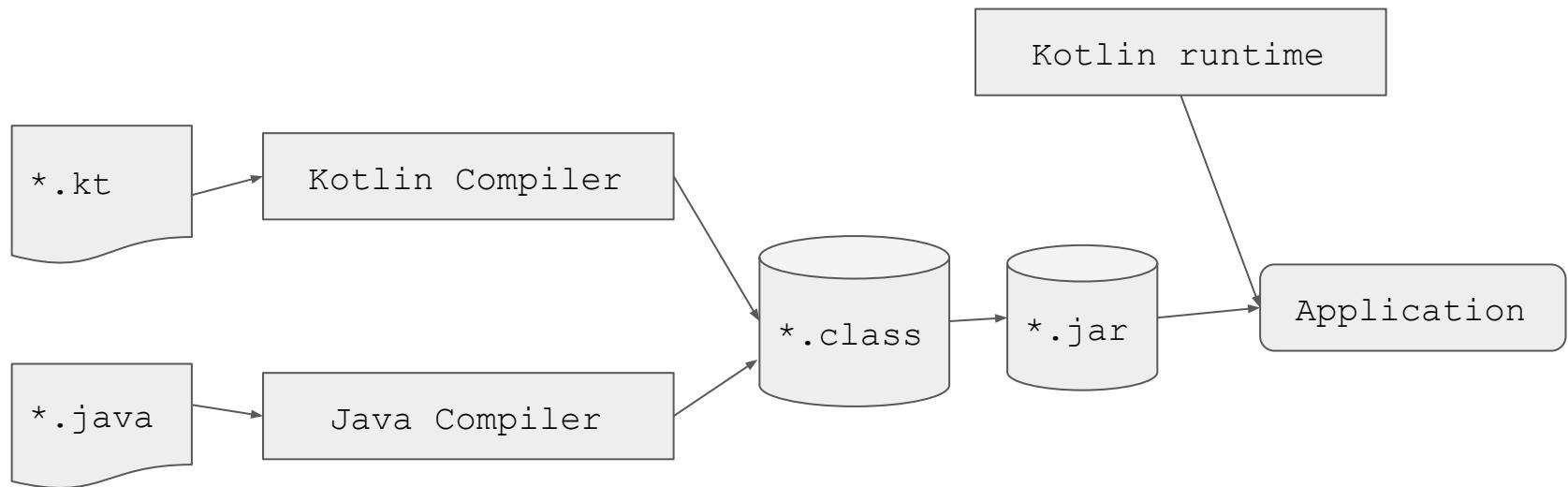
## Historia de Kotlin

- Creado en 2011 por **JetBrains**.
- Liberado en 2012 con licencia Apache 2.
- En 2016 sale la versión 1.0.
- En 2017 se anuncia en el Google I/O que Kotlin será un lenguaje oficial para desarrollar aplicaciones en Android.
- En 2019 Google anuncia que Kotlin es su lenguaje preferido para desarrollar aplicaciones en Android.

# Introducción a Kotlin

- Kotlin es un lenguaje tipado.
- Es Null Safety.
- Corre sobre la Java Virtual Machine.
- 100% interoperable con Java.
- Es un lenguaje de programación funcional.
- Es facil de aprender.

# Cómo funciona Kotlin?



# Hola Mundo en Kotlin

```
fun main() {  
    println("Hola Mundo")  
}
```

# Comentarios en Kotlin

```
// Este es un comentario de una linea.
```

```
/*
```

```
Este es un comentario de  
multiples lineas.
```

```
*/
```

# Variables en Kotlin

```
// 1. Variables Mutables
```

```
var puntaje = 100
```

```
puntaje = puntaje + 1
```

```
// 2. Variables Inmutables
```

```
val nombre = "Maria"
```

```
nombre = "Gloria"    // Error de compilacion
```

# Tipos de Datos en Kotlin (I)

```
// 1. String  
  
val nombre = "Ivete"  
  
val apellido: String = "Sangalo"  
  
  
// 2. Character  
  
val nota = 'A'  
  
val sexo: Char = 'F'  
  
  
// 3. Boolean  
  
val gano = true  
  
Val perdio: Boolean = false
```

# String templates

```
var a = 1

val s1 = "El valor de a es $a" // El valor de a es 1

a = 2

val s2 = "${s1.replace("es", "era")}, pero ahora es $a"
// El valor de a era 1, pero ahora es 2
```

## Valores Nulos

```
var edad = null      // Error de compilacion  
  
var id:String? = null // Correcto  
  
println(id.length)    // Error de compilacion  
  
println(id?.length)   // Correcto
```

## Operador Elvis ?:

```
val id:String? = null // Correcto
```

```
val tam = id?.length?:0
```

## Operador Double Bang !!

```
val id:String? = null
```

```
val tam = id!!.length //NullPointerException forzado
```

# Arreglos

```
val mixto = arrayOf(2, "tres", 'C', 7.0)  
println(mixto[0]) // Imprime 2  
  
val meses = arrayOf("Enero", "Febrero", "Marzo")  
println (meses[2]) // Imprime Marzo  
  
val edades:IntArray = intArrayOf(30, 25, 28, 32, 40)  
val suma = edades[0] + edades[1] // 55
```

# Conversión de Tipos de Datos

```
// Convertir un entero a un Long  
  
val codigo:Int = 123  
  
val codigoLargo:Long = codigo.toLong()  
  
// Otros métodos para convertir:  
  
val codigoB = codigo.toByte()  
  
val codigoS = codigo.toShort()  
  
val codigoI = codigo.toInt()  
  
val codigoL = codigo.toLong()  
  
val codigoF = codigo.toFloat()  
  
val codigoD = codigo.toDouble()
```

# Lectura de Datos de Teclado

```
fun main() {  
  
    println("=====")  
  
    println("===== BIENVENIDOS =====")  
  
    println("=====")  
  
    print("Ingresa tu Nombre: ")  
  
    val nombre = readLine()  
  
    print("Ingresa tu Edad: ")  
  
    val edad = readLine()?.toInt() ?: 0  
  
    println("Tu nombre es $nombre y tu edad es $edad")  
  
}
```

# Operadores Aritméticos

Expresión	Nombre de la función	Equivale a
$x + y$	plus	$x.\text{plus}(y)$
$x - y$	minus	$x.\text{minus}(y)$
$x * y$	times	$x.\text{times}(y)$
$x / y$	div	$x.\text{div}(y)$

# **Subprogramas**

# Funciones (I)

```
fun triplicar(numero: Int): Int {  
    return numero * 3  
}  
  
fun main() {  
    val numero = 4  
    val triple = triplicar(numero)  
    println("El triple de $numero es $triple")  
}
```

## Funciones (II)

```
fun triplicar(numero: Int) = numero * 3

fun main() {
    val numero = 4
    val triple = triplicar(numero)
    println("El triple de $numero es $triple")
}
```

# Procedimientos (I)

```
fun triplicar(numero: Int): Unit {  
    val triple = numero*3  
    println("El triple de $numero es $triple")  
}  
  
fun main() {  
    triplicar(4)  
}
```

## Procedimientos (II)

```
fun triplicar(numero: Int) {  
    val triple = numero*3  
    println("El triple de $numero es $triple")  
}  
  
fun main() {  
    triplicar(4)  
}
```

# **Estructuras de Control**

# Operadores Relacionales

Operador	Nombre
<code>==</code>	Igual a
<code>!=</code>	Distinto de
<code>&lt;</code>	Menor que
<code>&lt;=</code>	Menor o igual que
<code>&gt;</code>	Mayor que
<code>&gt;=</code>	Mayor o igual que

## Sentencia If, If - Else

```
// Sentencia If

val edad = readLine() !!.toInt()

if (edad > 18) {
    println ("Eres mayor de edad!")
}
```

```
// Sentencia If - Else

val compra = readLine() !!.toDouble()

if (compra >= 500) {
    descuento = 0.20
}
else {
    descuento = 0.10
}
```

# Operadores Lógicos: O

<b>Expresión Lógica: A</b>	<b>Expresión Lógica: B</b>	<b>A    B</b>
true	true	true
true	false	true
false	true	true
false	false	false

# Operadores Lógicos: Y

<b>Expresión Lógica: A</b>	<b>Expresión Lógica: B</b>	<b>A &amp;&amp; B</b>
true	true	true
true	false	false
false	true	false
false	false	false

# Operadores Lógicos: Negación

Expresión Lógica: A	$\neg A$
true	false
false	true

# Sentencia When

```
var literal = ""

val digito = readLine() !!.toInt()

when (digito) {
    0 -> literal = "cero"
    1 -> literal = "uno"
    2 -> literal = "dos"
    3 -> literal = "tres"
    4 -> literal = "cuatro"
    5 -> literal = "cinco"
    6 -> literal = "seis"
    7 -> literal = "siete"
    8 -> literal = "ocho"
    9 -> literal = "nueve"
    else -> println("No es un digito")
}
```

# Ciclo For

```
//Mostrar los numeros del 1 al 10
for (x in 1..10) {
    println(x)      // 1  2  3  4  5  6  7  8  9  10
}
```

```
//Del 1 hasta un numero leido por teclado
val limite = readLine()!!.toInt()
for (x in 1..limite) {
    println(x)
}
```

```
//Mostrar los valores de un arreglo
val numeros = arrayOf(10, 20, 30, 40, 50)
for (indice in 0..numeros.size-1) {
    println(numeros[indice])
}
```

# Ciclo While

```
//Mostrar los numeros del 1 al 10
var x = 1
while (x <= 10) {
    println(x)
    x++
}
```

# Ciclo Do-While

```
//Mostrar los numeros del 1 al 10
var x = 1
do {
    println(x)
    x++
}while (x <= 10)
```

# **Programación Orientada a Objetos**

# Conceptos de Orientación a Objetos

- **Clases:** patrones que indican como construir objetos.
- **Objetos:** instancias de las clases en tiempo de ejecución.

Ejemplo: plano de arquitecto vs edificios

- Miembros de la clase:
  - ◆ Propiedades: características variables de los objetos. Pueden ser variables mutables o inmutables.
  - ◆ Métodos: comportamiento de los objetos. Son funciones que operan sobre los atributos de los objetos.

# Hola Mundo Orientado a Objetos en Kotlin

```
class Greeter(val name: String) {  
    fun greet() {  
        println("Hello, $name")  
    }  
}  
  
fun main() {  
    val name = readLine()  
    Greeter(name).greet() // Hello, Anonymous  
}
```

# Clases en Kotlin

```
// Crear una clase
```

```
class Invoice { }
```

```
// Si no hay ni métodos ni atributos:
```

```
class Invoice
```

# Constructores

```
// Existe un constructor primario

class Person constructor(firstName: String) { }

// La palabra reservada constructor puede ser omitida

class Person (firstName: String) { }

// Puede haber uno o más constructores secundarios

class Person {

    constructor (lastName: String) {

    }
}
```

# Creando instancias

```
val invoice = Invoice()  
val customer = Customer("Juan Perez")
```

## Ejemplo de creación de una Clase

```
class Date {  
  
    var day: Int? = null  
  
    var month: Int? = null  
  
    var year: Int? = null  
  
    constructor (d: Int, m: Int, y: Int) {  
  
        this.day = d  
  
        this.month = m  
  
        this.year = y  
  
    }  
  
}
```

## Ejemplo de creación de un Objeto

```
fun main() {  
  
    val myBirthDay:Date = Date(24, 7, 1983)  
  
    println("My birthday is on  
${myBirthDay.month}/${myBirthDay.day}")  
  
}
```

# Herencia

```
// Herencia implicita

class Example { } // Hereda por defecto de Any

// Para permitir herencia de clases

open class Animal(p: Int) { }

class Gato(p: Int) : Base(p) { }
```

# Sobreescritura de Métodos

```
open class Shape {  
    open fun draw() { /*...*/ }  
  
    fun fill() { /*...*/ }  
}  
  
class Circle() : Shape() {  
    override fun draw() { /*...*/ }  
}
```

# Ejemplo de Herencia

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(state: Bundle?) {  
  
        super.onCreate(state)  
  
        setContentView(R.layout.activity_main)  
  
    }  
  
}
```

# **Introducción al desarrollo de Aplicaciones para Android**

<https://developer.android.com>

# Android Apps tienen múltiples puntos de Entrada

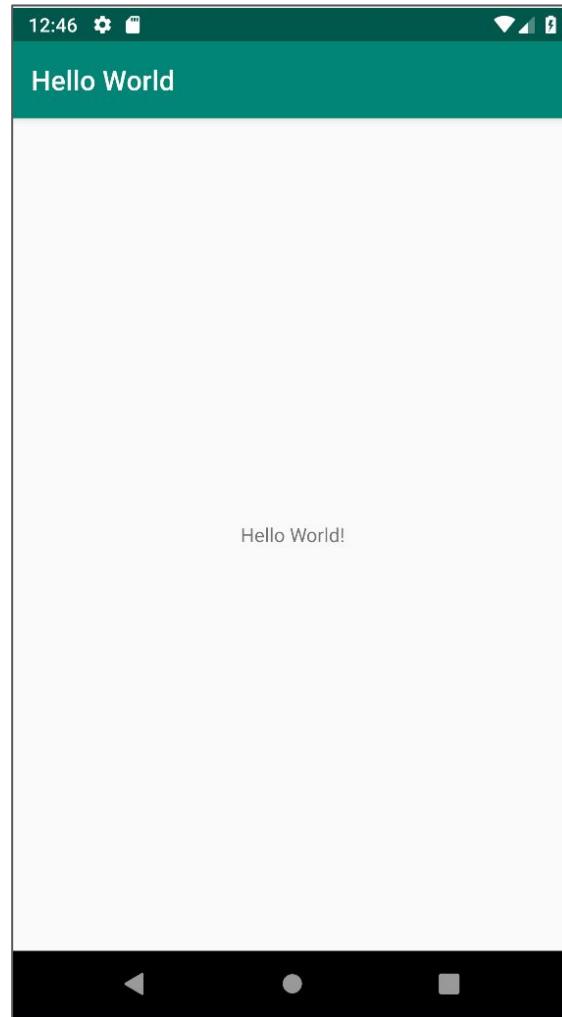
- Android App = Conjunto de componentes que pueden ser invocados individualmente.
  - ◆ Ej: Una Actividad (Activity), es un tipo de componente que provee una interfaz de usuario.
- La Actividad principal se puede iniciar desde el icono de la aplicación desde el escritorio o desde una notificación o desde otra aplicación diferente.
- Otros componentes como Broadcast Receivers y Services permiten realizar tareas en background, sin interfaz de usuario.

# Android Apps se adaptan a diferentes dispositivos

- Android permite proveer diferentes recursos para diferentes dispositivos.
  - ◆ Por ejemplo: Se puede crear diferente diseños para diferentes tamaños de pantalla. El sistema determinar que diseño utilizar basado en el tamaño de pantalla del dispositivo actual.
- Si alguna funcionalidad de la aplicación requiere de algún hardware específico como una cámara, se puede consultar en tiempo de ejecución si el dispositivo posee ese hardware o no y deshabilitar la funcionalidad si es que no la tiene.
- Se puede especificar en Google Play que la aplicación requiere algún hardware específico para permitir la instalación.

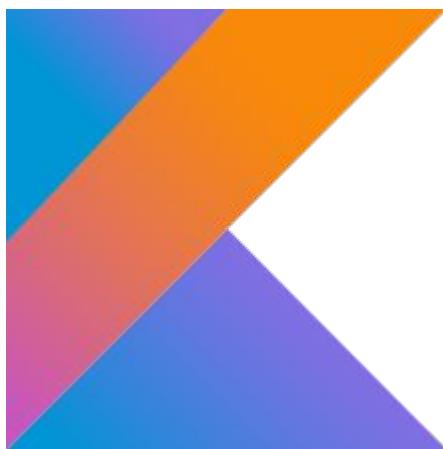
# Codelab #2: Hello World!

<https://bit.ly/38cGNWz>

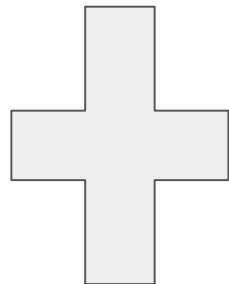


# Activity

→ Kotlin + XML



Activity



Layout

# Recursos de la aplicación

- Son archivos adicionales y contenido estático que usa el código.  
Ej: bitmaps, diseños, cadenas de texto, instrucciones de animación, etc.
- La carpeta `res/` contiene todos los recursos de la aplicación.

```
MyProject/
  src/
    MyActivity.kt
  res/
    drawable/
      graphic.png
    layout/
      main.xml
      info.xml
    mipmap/
      icon.png
    values/
      strings.xml
```

<https://developer.android.com/guide/topics/resources/providing-resources>

# **Widgets (Views)**

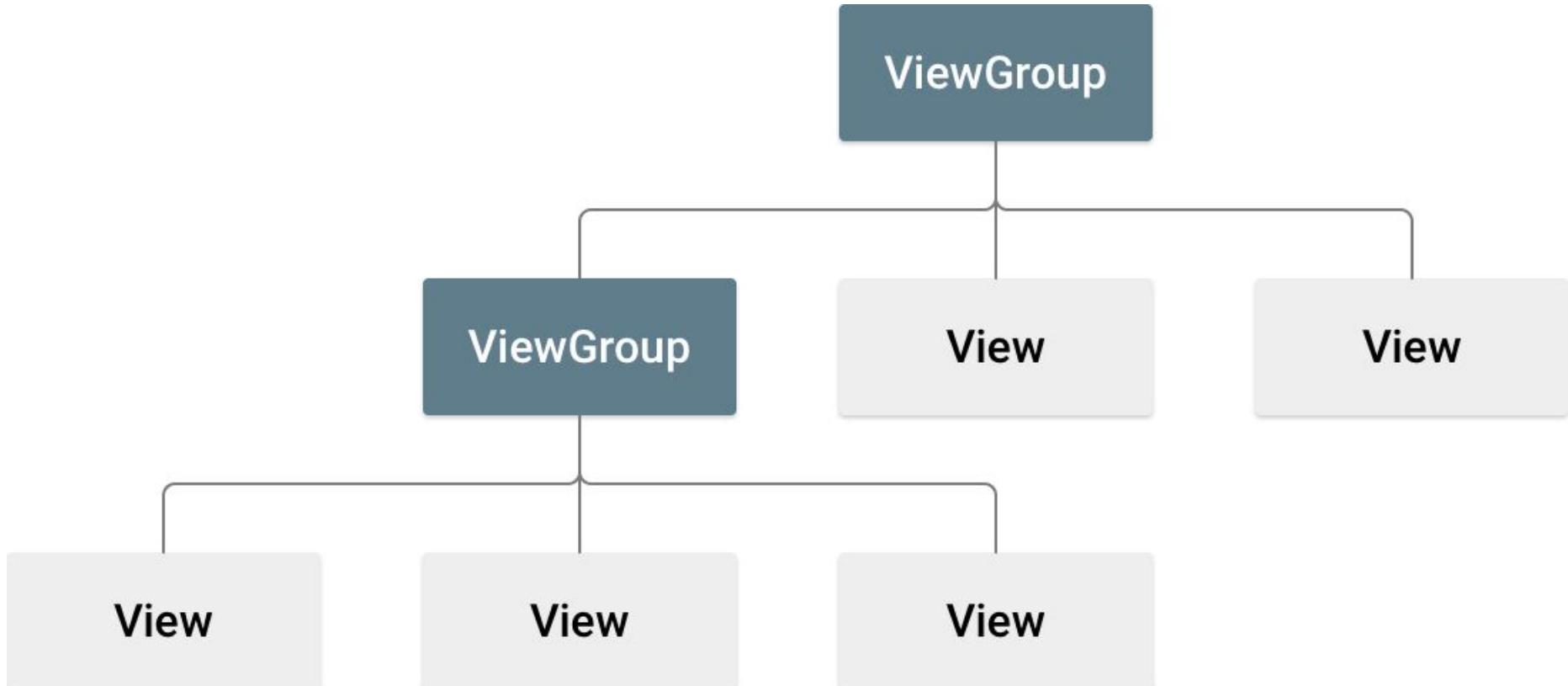
**View:** Es un objeto que sirve para dibujar algo en pantalla con el que el usuario puede interactuar.

**ViewGroup:** Es un objeto que se compone de otros views ordenados.  
(Layout)

# **TextView**

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!" />
```

# Layouts



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

# **Layouts**

Hay 2 formas de declarar layouts:

1. XML
2. Kotlin

# Layout XML

```
<?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>
```

# Cargando el Layout XML

```
[ . . . ]  
  
fun onCreate(savedInstanceState: Bundle) {  
  
    super.onCreate(savedInstanceState)  
  
    setContentView(R.layout.main_layout)  
  
}  
  
[ . . . ]
```

# Atributos

**Id** -> Identificador de la vista

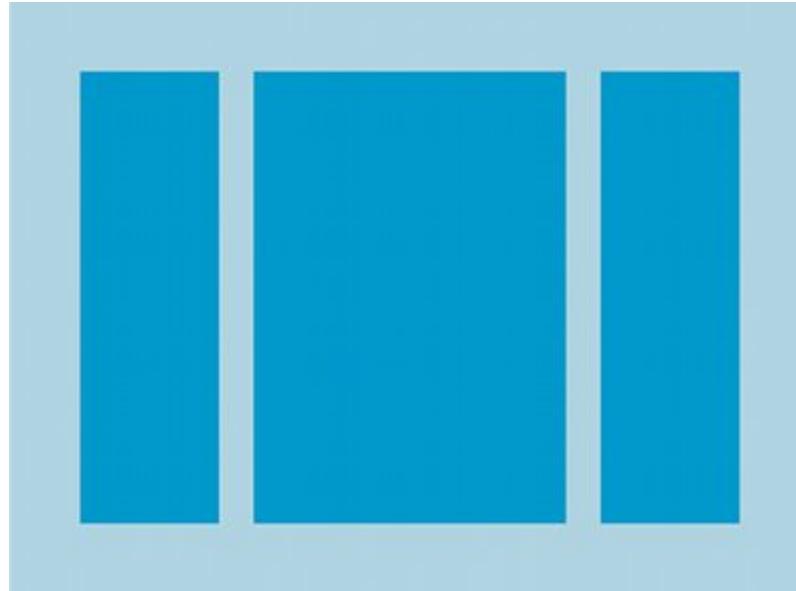
Ejemplo:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text" />
```

Para obtener una referencia desde codigo:

```
my_button = findViewById(R.id.my_button)
o
import kotlinx.android.synthetic.main.activity_main.*
[...]
my_button.text = "Aceptar"
```

# **LinearLayout**



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

# Codelab #3: Anatomy of Basic Android Project

<https://bit.ly/2JvNulX>

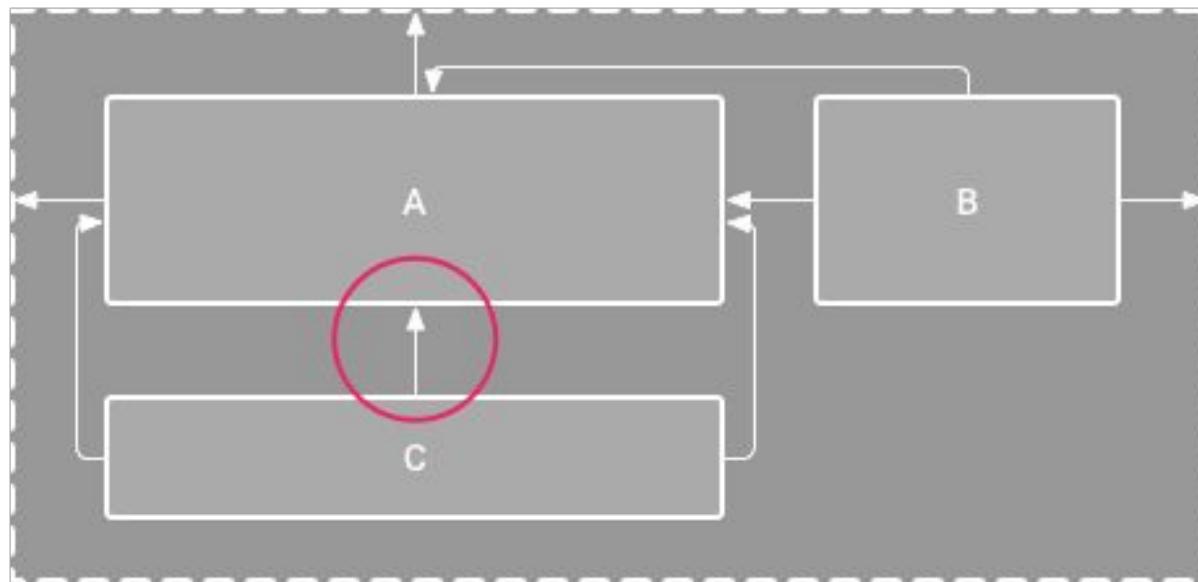


# RelativeLayout



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

# ConstraintLayout



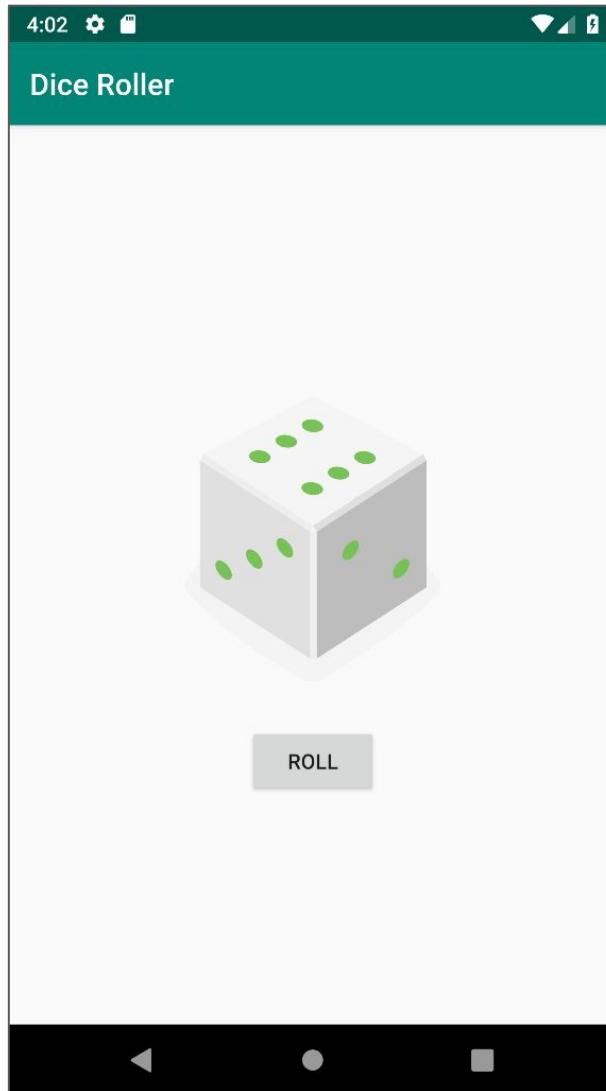
<https://developer.android.com/training/constraint-layout/index.html>

## API levels

1. [https://medium.com/androiddevelopers/picking-your-compilesdkve  
rsion-minsdkversion-targetsdkversion-a098a0341ebd#.bpqsg4ili](https://medium.com/androiddevelopers/picking-your-compilesdkversion-minsdkversion-targetsdkversion-a098a0341ebd#.bpqsg4ili)

# Codelab #4: Images Resources and compatibility

<https://bit.ly/3mVRr8f>

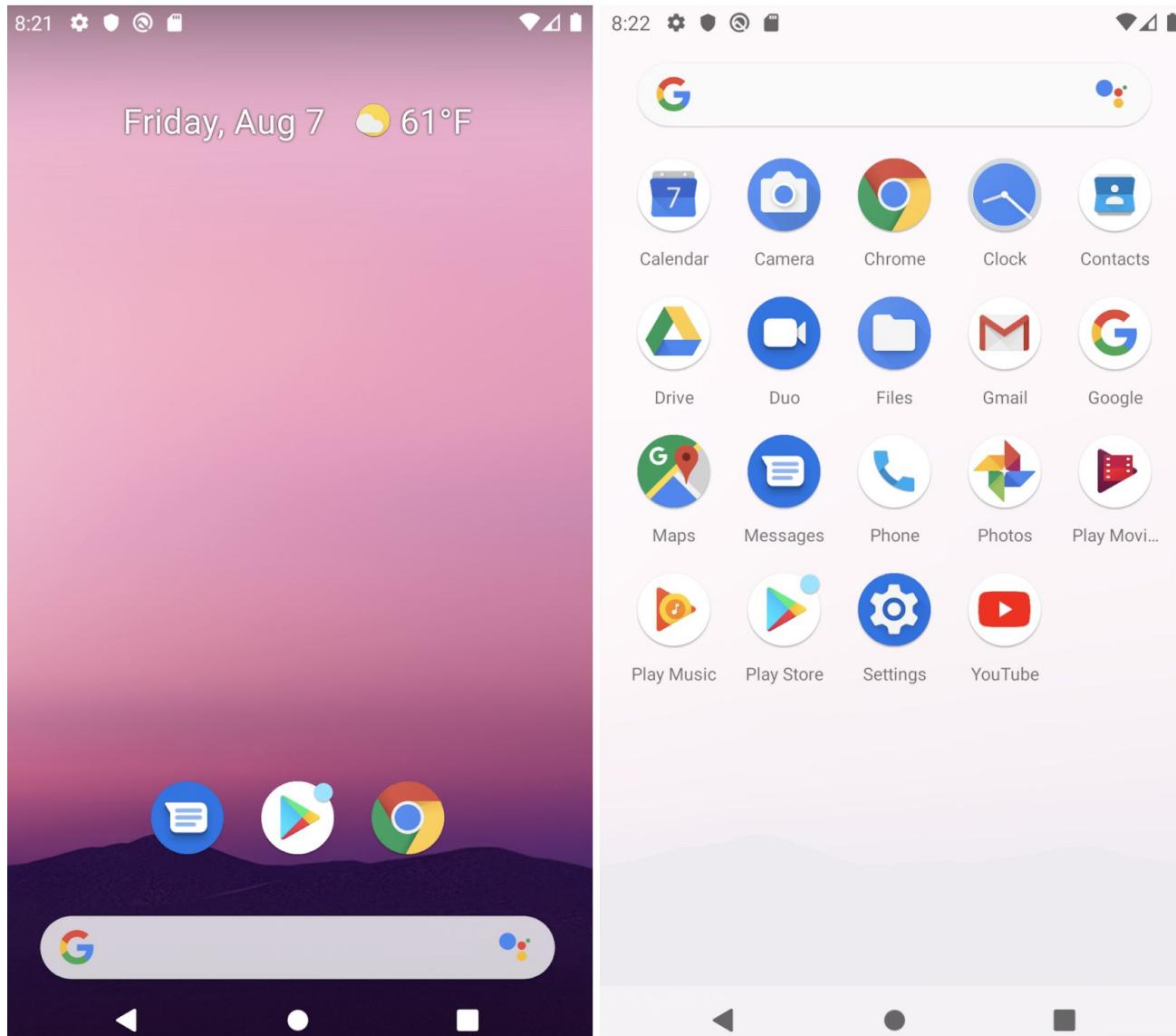


[Starting App](#)

## Icono de la aplicación (I)

- El icono de aplicación es una manera muy importante de distinguir nuestra aplicación.
- Aparece también en diferentes lugares como en la pantalla principal o escritorio, en configuraciones, etc.
- También se conoce como **Launcher Icon**, que sería lanzador de la aplicación que es la experiencia cuando hacemos clic sobre el ícono para levantar el Activity principal de la aplicación.

## Icono de la aplicación (II)

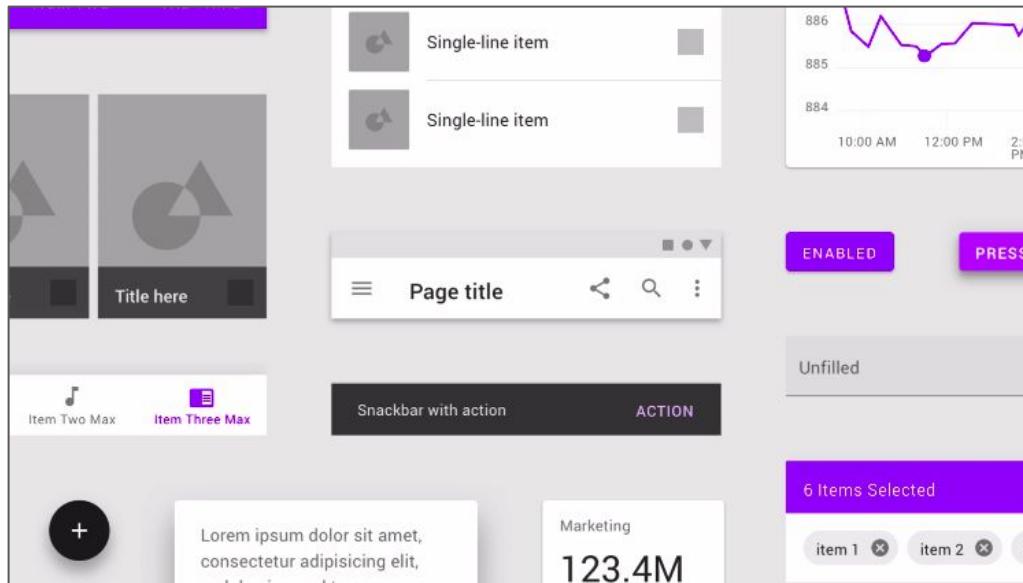


# **Codelab # 5: Learn to help yourself**

<https://bit.ly/2Idy1fS>

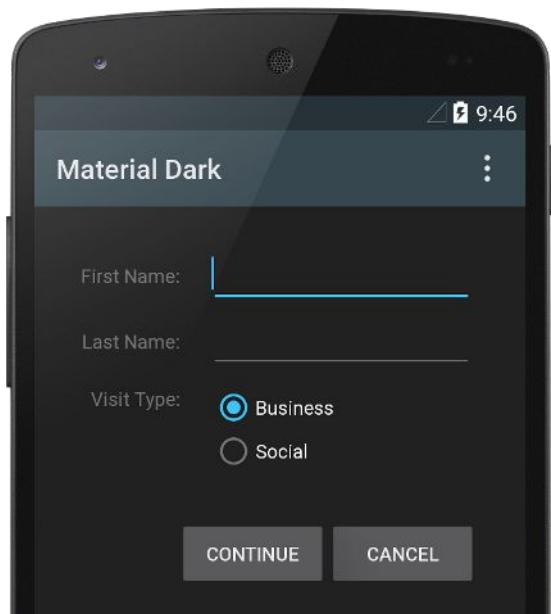
# Material Design

- Es una guía completa para el diseño visual, interactivo y de movimiento en diferentes plataformas y dispositivos.
- Son métricas de diseño creadas por Google para dispositivos móviles y para la web.



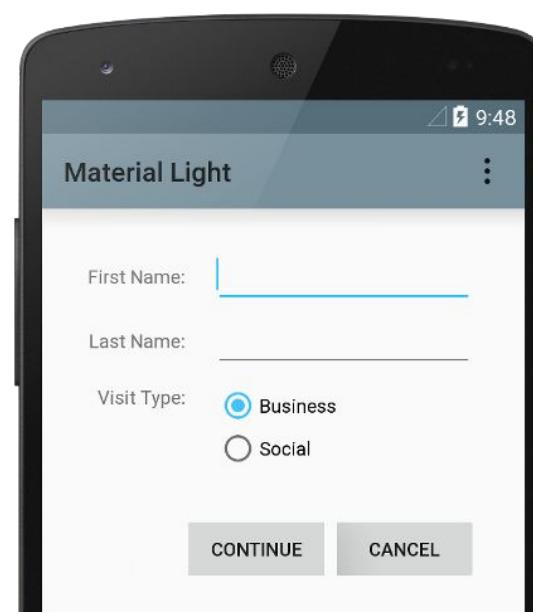
# Material Design en Android

Archivo: res/values/styles.xml



Tema Oscuro

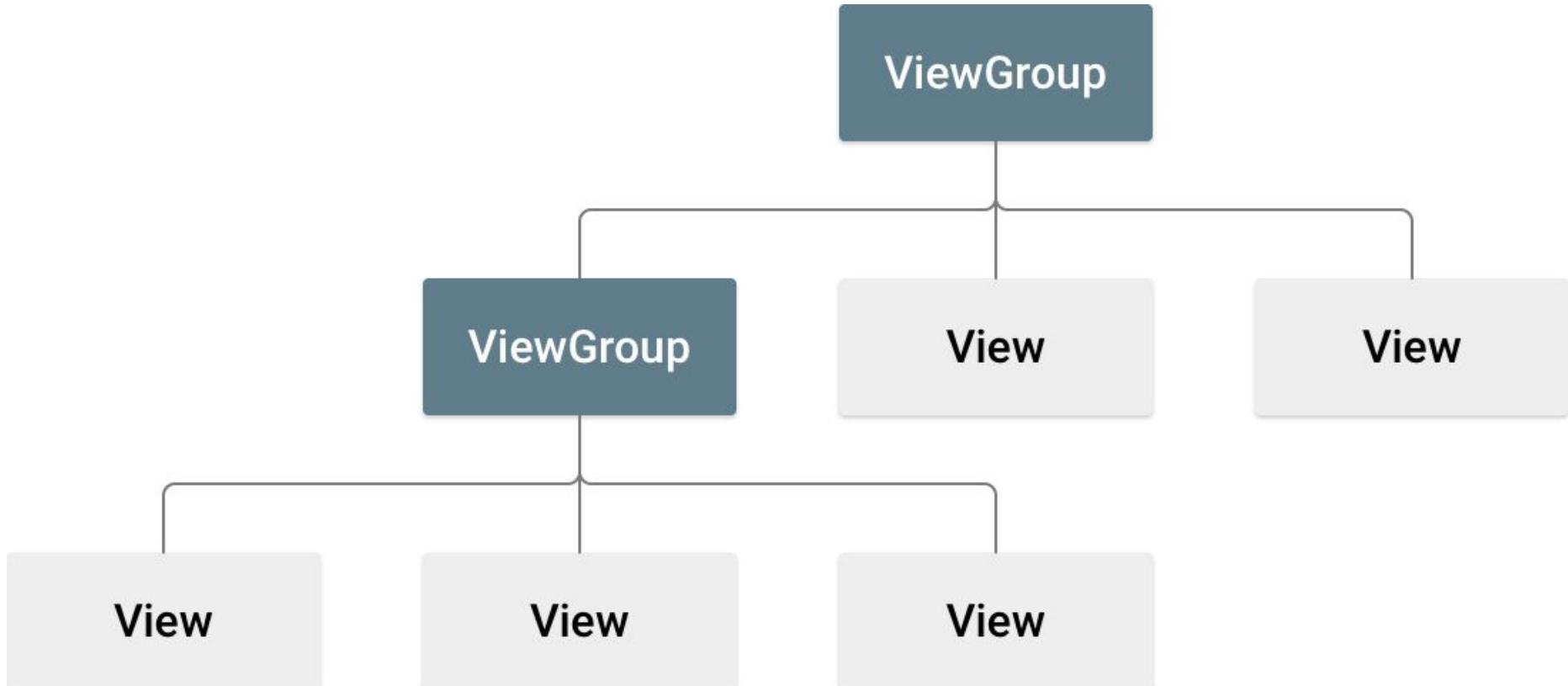
Theme.AppCompat



Tema Claro

Theme.AppCompat.Light

# Layouts



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

# **Codelab # 6: LinearLayout using Layout Editor**

<https://bit.ly/2U3sUBG>

# **Codelab # 7: User Interactivity**

<https://bit.ly/3nd26vm>

# **Codelab # 8: ConstraintLayout using Layout Editor**

<https://bit.ly/2GOPawn>

## Kotlin Features - lambda expression

- Son funciones anónimas.
- Ejemplos:

```
{println("Hola Mundo")} ()  
  
var hola = {println("Hola Mundo")} ()  
  
hola() //Hola Mundo
```

```
val s1 = { a, b -> a + b}  
val s2 = { a: Int, b: Int -> a + b}  
println (s1(3,4))
```

```
doneButton.setOnClickListener {  
    addNickname(it)  
}
```

## Kotlin Features - lateinit

- lateinit permite inicializar los atributos en métodos diferentes al método constructor.

```
// Sin lateinit
class Person {
    private var name: String //Error de compilación

    fun setNombre(name: String) {
        this.name = name
    }
}

// Con lateinit
class Person {
    private lateinit var name: String // Todo bien :)

    fun setNombre(name: String) {
        this.name = name
    }
}
```

## Kotlin Features - apply

- Una función que recibe y devuelve this, el contexto del objeto que hace la llamada.
- Se resume como aplicar las siguientes asignaciones al objeto. Ej:

```
val adam = Person("Adam") .apply {  
    age = 32  
    city = "London"  
}  
println(adam)
```

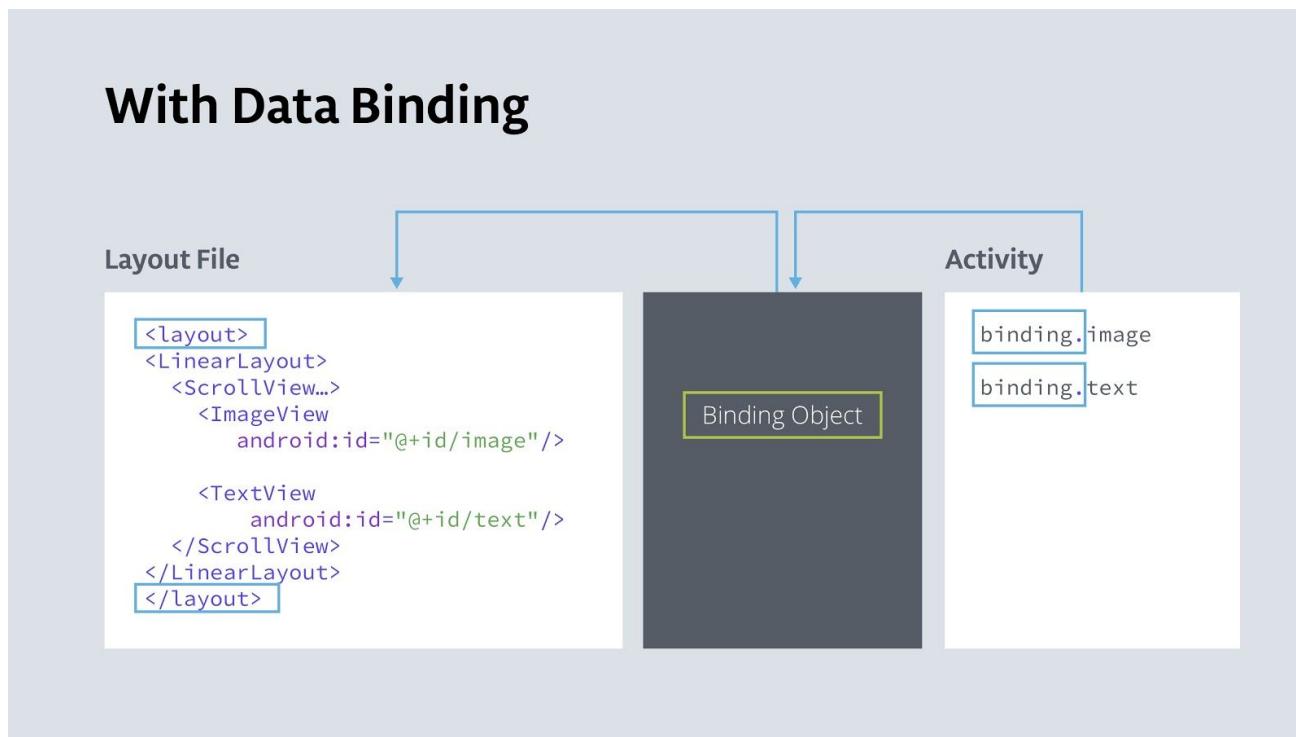
## Kotlin Features - data class

- Una clase destinada principalmente a guardar datos:

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

# Data Binding

- Una alternativa mucho más eficiente al enlace entre un widget del layout y su referencia objeto en un activity.
- Mejor separación entre la vista (layout) y los datos.



# Codelab # 9: Data Binding Basics

<https://bit.ly/3ljXPpy>

About Me

Aleks Haecky

What is your Nickname?

DONE

★

Hi, my name is Aleks.  
I love fish.  
The kind that is alive and swims around  
in an aquarium or river, or a lake, and  
definitely the ocean.  
Fun fact is that I have several aquariums  
and also a river.  
I like eating fish, too. Raw fish. Grilled  
fish. Smoked fish. Poached fish – not so  
much.  
And sometimes I even go fishing.  
And even less sometimes. I actually

About Me

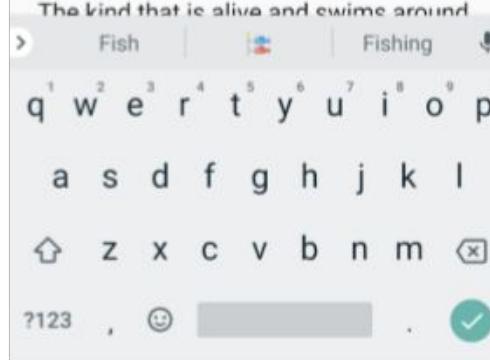
Aleks Haecky

Kotlin Fish

DONE

★

Hi, my name is Aleks.  
I love fish.  
The kind that is alive and swims around  
in an aquarium or river, or a lake, and  
definitely the ocean.  
Fun fact is that I have several aquariums  
and also a river.  
I like eating fish, too. Raw fish. Grilled  
fish. Smoked fish. Poached fish – not so  
much.  
And sometimes I even go fishing.  
And even less sometimes, I actually  
catch something



About Me

Aleks Haecky

Kotlin Fish

★

Hi, my name is Aleks.  
I love fish.  
The kind that is alive and swims around  
in an aquarium or river, or a lake, and  
definitely the ocean.  
Fun fact is that I have several aquariums  
and also a river.  
I like eating fish, too. Raw fish. Grilled  
fish. Smoked fish. Poached fish – not so  
much.  
And sometimes I even go fishing.  
And even less sometimes, I actually  
catch something

Once, when I was camping in Canada,

## Kotlin Features - singleton objects

- Una clase que solo puede tener una instancia:

```
object CarFactory {  
    val cars = mutableListOf<Car>()  
  
    fun makeCar(horsepowers: Int): Car {  
        val car = Car(horsepowers)  
        cars.add(car)  
        return car  
    }  
}  
  
[ ... ]  
  
val car = CarFactory.makeCar(150)  
println(CarFactory.cars.size)
```

## Kotlin Features - companion objects

- El equivalente a los métodos o atributos estáticos en Java:

```
class Car(val horsepowers: Int) {  
    companion object Factory {  
        val cars = mutableListOf<Car>()  
  
        fun makeCar(horsepowers: Int): Car {  
            val car = Car(horsepowers)  
            cars.add(car)  
            return car  
        }  
    }  
}  
  
[ ... ]  
  
val car = Car.makeCar(150)  
println(Car.Factory.cars.size)
```

## Fragments (I)

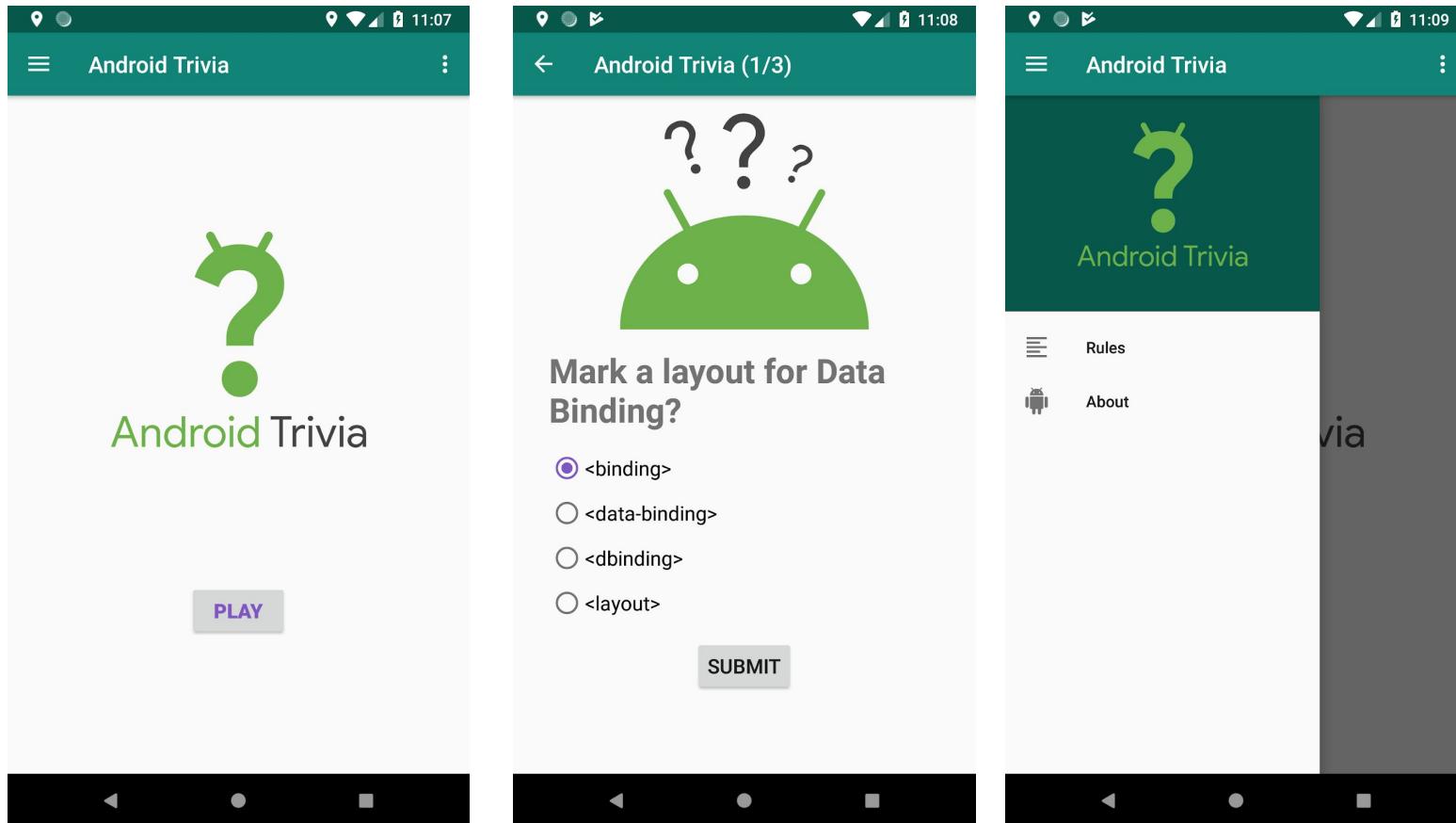
- Representa un fragmento o sección de un activity.
- Se puede combinar varios fragments en una sola actividad para construir una experiencia de usuario multipanel.
- Se puede reutilizar un fragment en múltiples activities.
- Podemos ver a un fragment como un sub-activity.

## Fragments (II)

- Un Fragment es similar a un Activity.
- Tiene su propio ciclo de vida y recibe eventos.
- Se puede añadir y quitar fragments en tiempo de ejecución.
- En Kotlin tenemos la clase base Fragment.
- El diseño de un fragment también se define en un archivo XML.

# Codelab #10: Fragments

<https://bit.ly/3puNgIO>



Starting App

## Android Jetpack

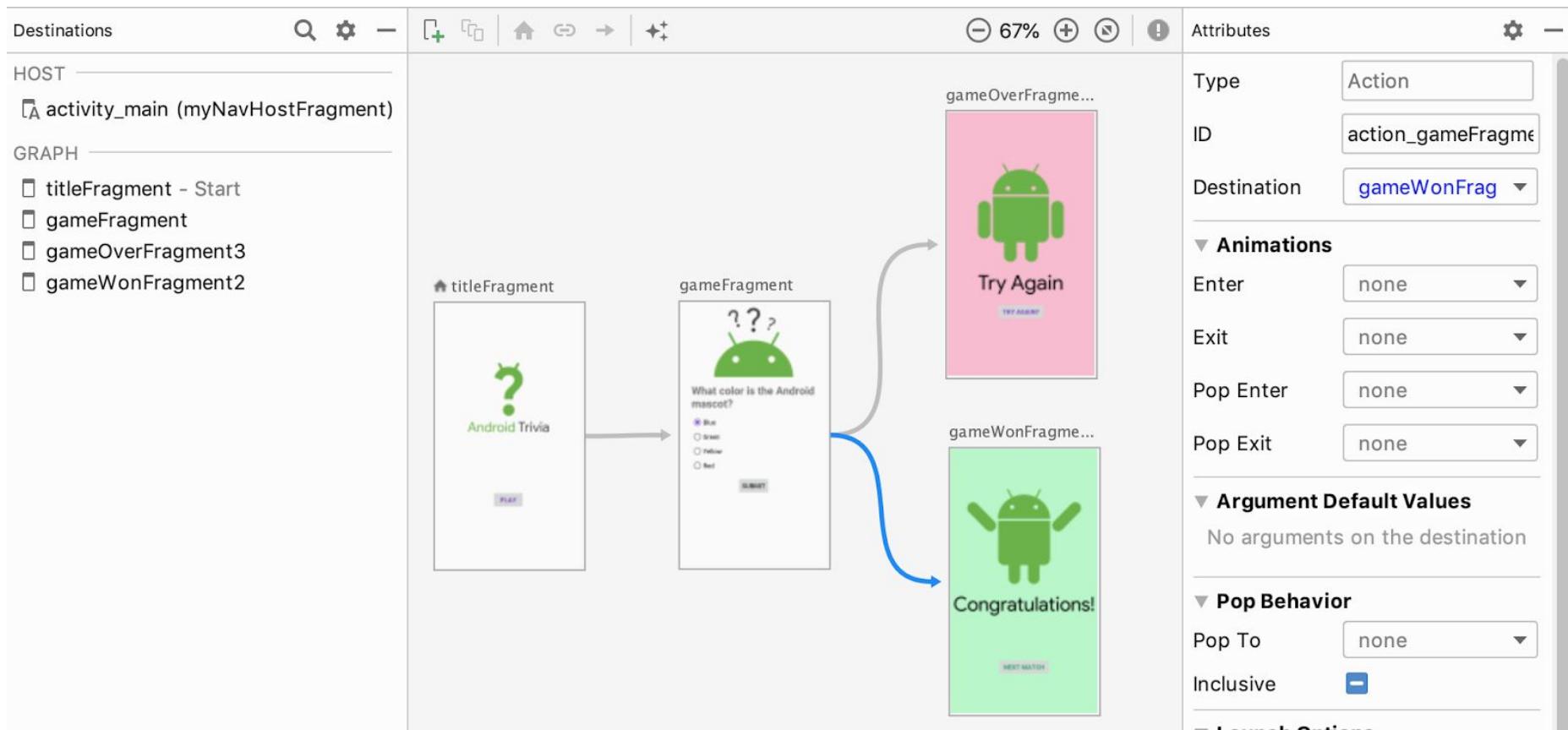
- Jetpack es un conjunto de librerías que ayuda a los desarrolladores a seguir las prácticas recomendadas, reducir el código estándar y escribir código que funcione de manera coherente en los dispositivos y las versiones de Android para que puedan enfocarse en el código que les interesa.



## Navigation (I)

- Se refiere a las interacciones que permiten a los usuarios navegar a través, dentro y fuera de las diferentes piezas de contenido de tu app.
- Es de **Android Jetpack** y permite implementar la navegación, desde simples clics de botones hasta patrones más complejos, como las barras de apps y los paneles laterales de navegación.
- También garantiza una experiencia del usuario coherente y predecible, ya que se adhiere a un sistema establecido de conjunto de principios.

# Navigation (II)



# Codelab #11: Navigation

<https://bit.ly/2H9OGjZ>

[Starting App](#)

# Intents

- Un Intent es un tipo de mensaje que se utiliza para hacer una solicitud desde un componente a otro.
- Algunos casos de uso:
  - ◆ Iniciar una Actividad
  - ◆ Iniciar un Servicio
  - ◆ Transmitir una Emisión

## Tipos de Intents

- Existen 2 tipos de intents:
  - ◆ **Explícitos:** Se especifica qué aplicación la realizará.
  - ◆ **Implícitos:** Se declara una acción general para que un componente de otra aplicación la maneje.

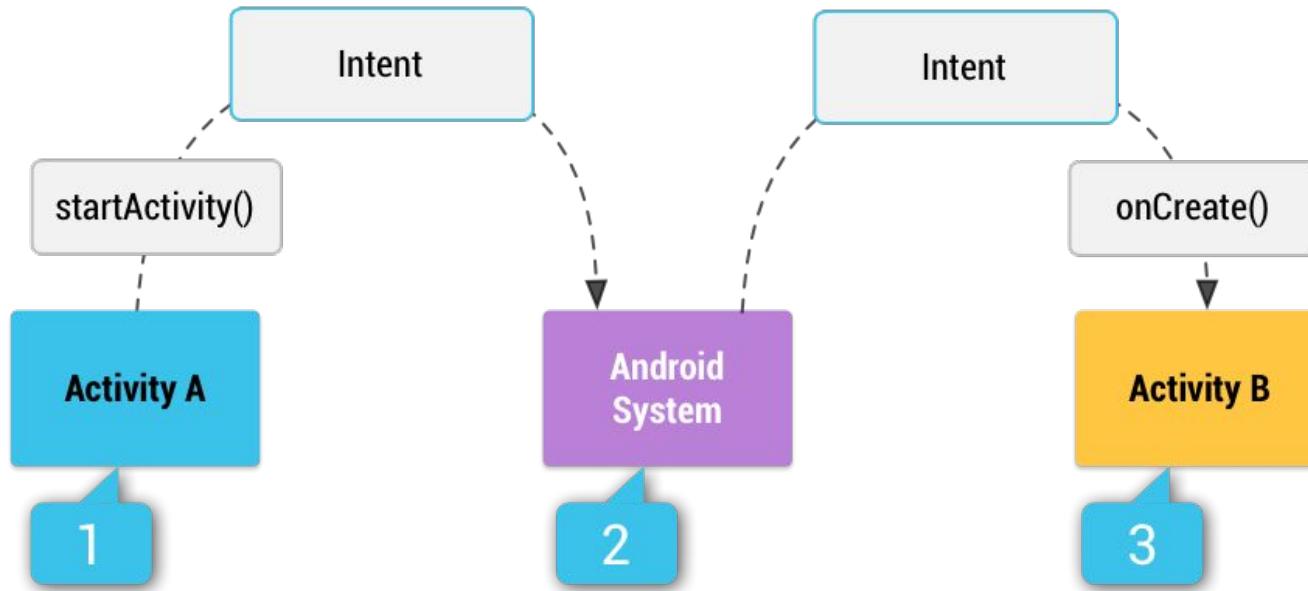
# Ejemplo de Intent Explícito

[ ... ]

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(downloadIntent)
```

[ ... ]

# Intent Implícito



- [1] Activity A crea un **Intent** con un action description y se lo pasa a **startActivity()**.
- [2] El sistema Android busca en todos los intent-filters de todas las apps y muestra las que cumplen al usuario.
- [3] Una vez el usuario selecciona una App el sistema inicia la actividad de la app (Activity B) llamando al método **onCreate()** y pasandole el **Intent**

# Ejemplo de Intent Implícito

[ ... ]

```
// Create the text message with a string  
val sendIntent = Intent().apply {  
    action = Intent.ACTION_SEND  
    putExtra(Intent.EXTRA_TEXT, textMessage)  
    type = "text/plain"  
}  
  
// Verify that the intent will resolve to an activity  
if (sendIntent.resolveActivity(packageManager) != null) {  
    startActivity(sendIntent)  
}
```

[ ... ]



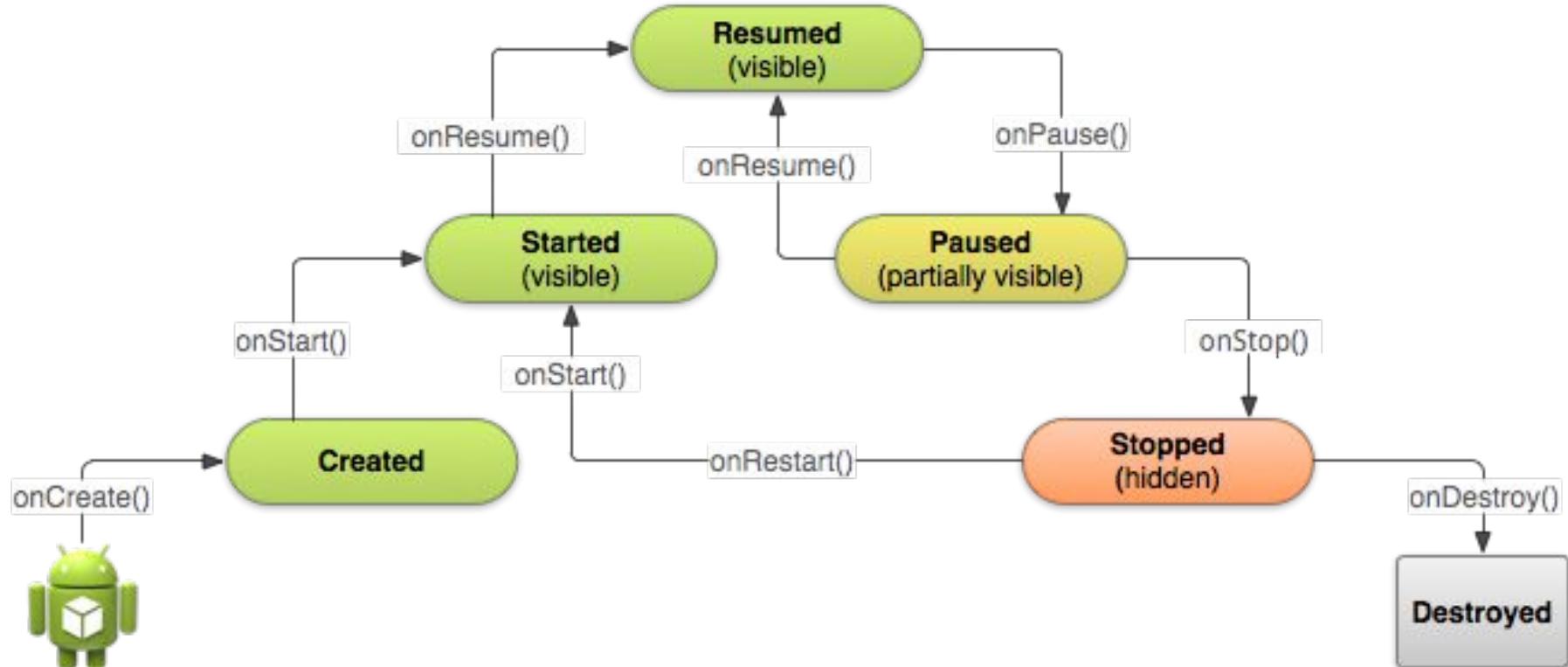
Se muestra al usuario todas las apps que pueden realizar esta intención de acción.

# Codelab #12: Start an external Activity

<https://bit.ly/3f8lrdb>

[Starting App](#)

# Activity Lifecycle



# Logging (I)

## Logging usando Log

- La clase `Log` del API de logging de Android habilita escribir mensajes de logs que son visibles en el `Logcat` de Android Studio.
- Por ejemplo `Log.i()` permite escribir mensaje de tipo informe.
- El panel `Logcat` de Android Studio permite ver todos los mensajes de logs del sistema incluidos los logs que añadimos.

# Logging (II)

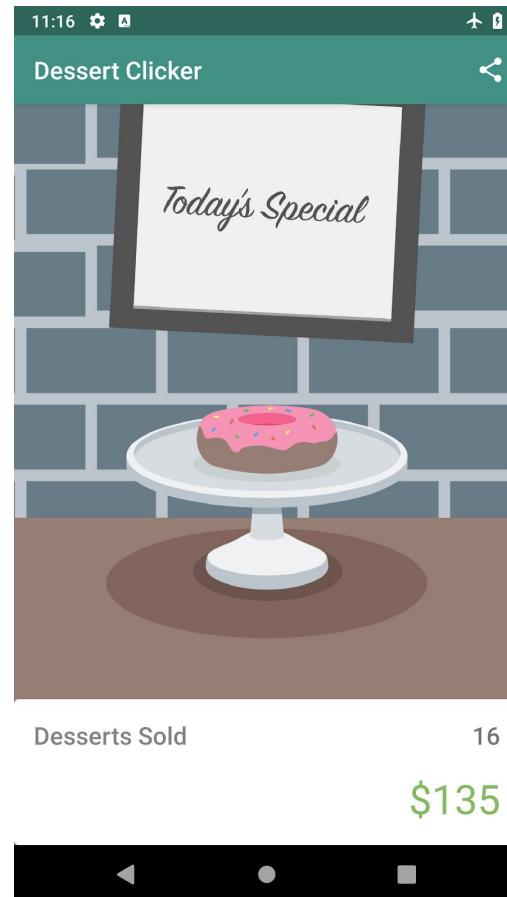
## Logging usando Timber

[Timber](#) es una librería para hacer logging con más ventajas sobre el API de logging por defecto de Android:

- Genera el log tag basado en el nombre de la clase.
- Ayuda a no mostrar los logs en la versión release de la app.
- Permite la integración con librerías de reporte de crashes.

# Codelab #13: Lifecycles and logging

<https://bit.ly/38QyT5F>



[Starting App](https://bit.ly/38QyT5F)

## Librería lifecycle

- Android Jetpack viene con la librería lifecycle para simplificar el correcto manejo de ciclo de vida de Activities y Fragments.
- Lifecycle tiene 3 partes:
  1. **Lifecycle owners** que son los activities y fragments.
  2. **La clase Lifecycle** que guarda el estado del ciclo de vida de los Lifecycle owners y lanza eventos cuando cambios ocurren.
  3. **Lifecycle observers** que son los que observan el ciclo de vida y realizan alguna tarea cuando hay un cambio en el ciclo de vida.

# Codelab #14: Complex Lifecycle Situations

<https://bit.ly/35CKIdG>

[Starting App](#)

# Android Jetpack

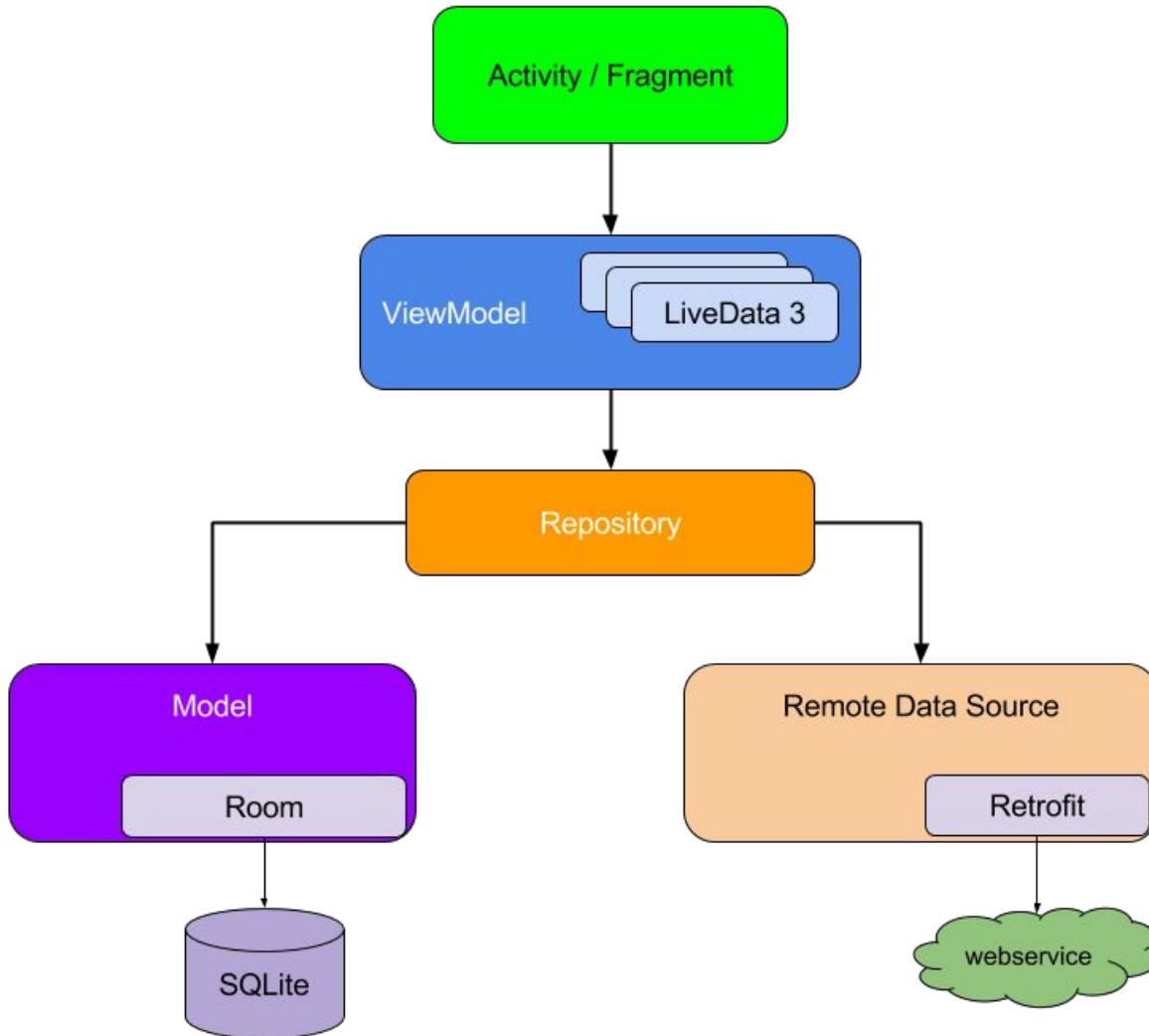
<https://developer.android.com/jetpack>

## Android Jetpack (I)

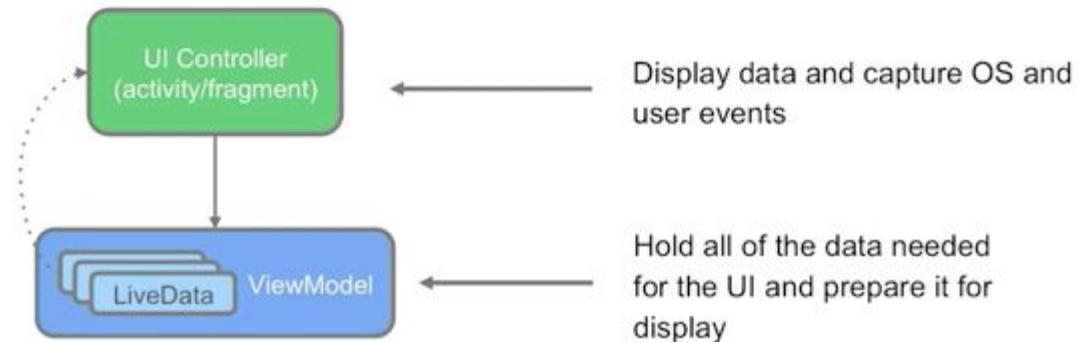
- Jetpack es un **conjunto de librerías** que ayuda a los desarrolladores a seguir las **prácticas recomendadas**, reducir el código estándar y escribir código que funcione de manera coherente en los dispositivos y las versiones de Android para que puedan enfocarse en el código que les interesa.



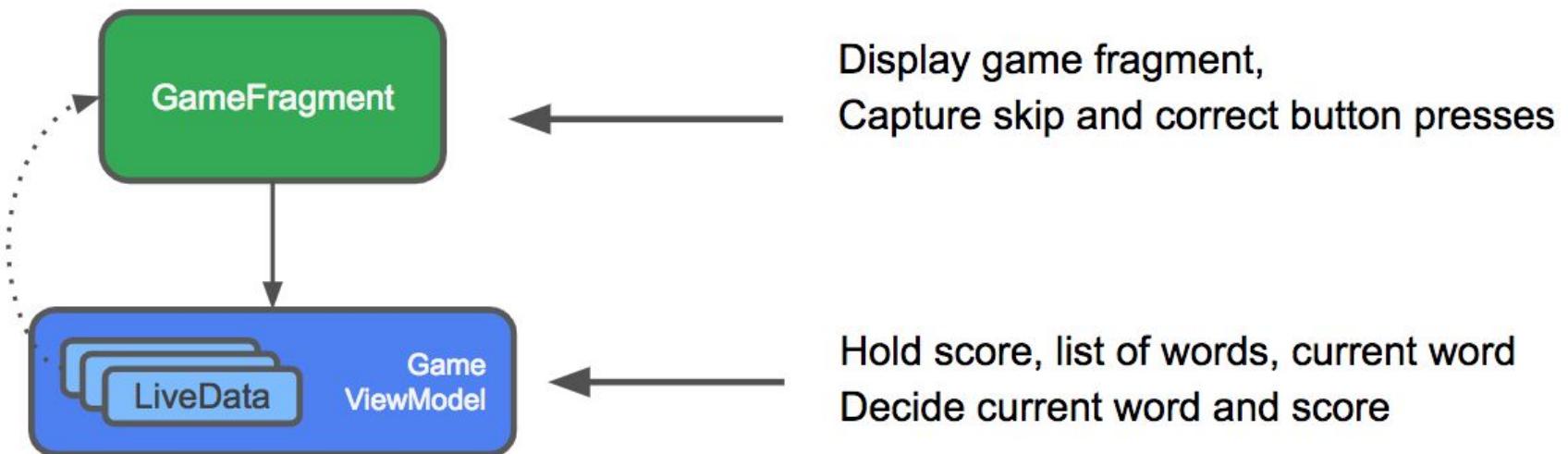
# Android Jetpack - Arquitectura



# ViewModel



# Ejemplo de ViewModel

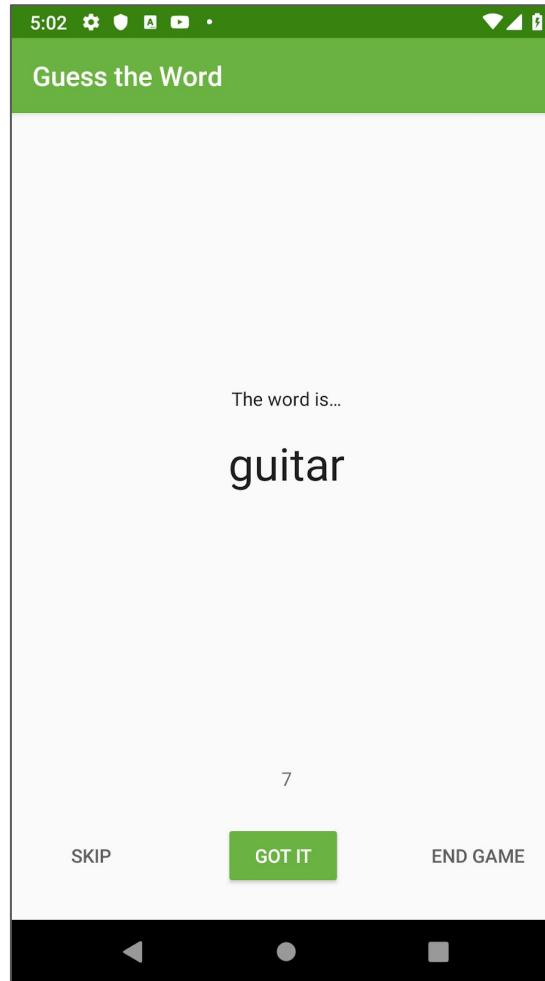


# **UI Controller vs ViewModel**

<b>UI Controller</b>	<b>ViewModel</b>
No debe contener datos para mostrar en el UI.	Contiene los datos que van a mostrarse en el UI.

# Codelab #15: View Model

<https://bit.ly/2UP925D>



[Starting App](https://bit.ly/2UP925D)

# LiveData

- Data que notifica a las vistas cuando hay cambios básicos de base de datos.
- Es un wrapper observable que notifica a sus observers (LiveData Observers) Activities o Fragments sobre cambios en los datos

## Kotlin Features - Backing properties

```
[...]
private var _table: Map<String, Int>? = null
public val table: Map<String, Int>
    get() {
        if (_table == null) {
            _table = HashMap()
        }
        return _table ?: throw AssertionError("Set
to null by another thread")
    }
[...]
```

# **Codelab #16: LiveData and LiveData Observers**

<https://bit.ly/393rAru>

[Starting App](#)

# Codelab #17: Data binding with ViewModel and LiveData

<https://bit.ly/3flkJKY>

[Starting App](#)

# Codelab #18: LiveData transformations

<https://bit.ly/3pLR3eZ>

[Starting App](#)

# **App Data and Files**

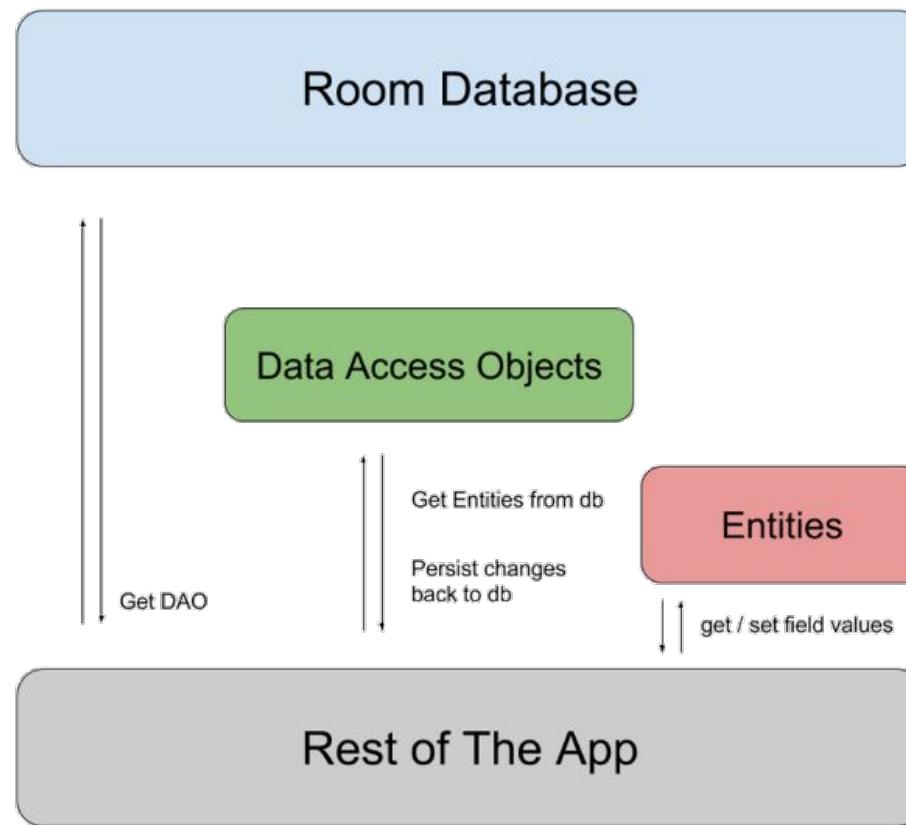
<https://developer.android.com/guide/topics/data>

## Opciones de Almacenamiento

- SharedPreferences: Key-Values
- Internal Storage: Private Data.
- External Storage: Public Data
- Network Connection: Storing on the web
- SQLite Database (**Room**): Storing structured data in a private DB.

# Room

- Room proporciona una capa de abstracción sobre SQLite que permite acceder a la base de datos sin problemas y al mismo tiempo aprovechar toda la potencia de SQLite.



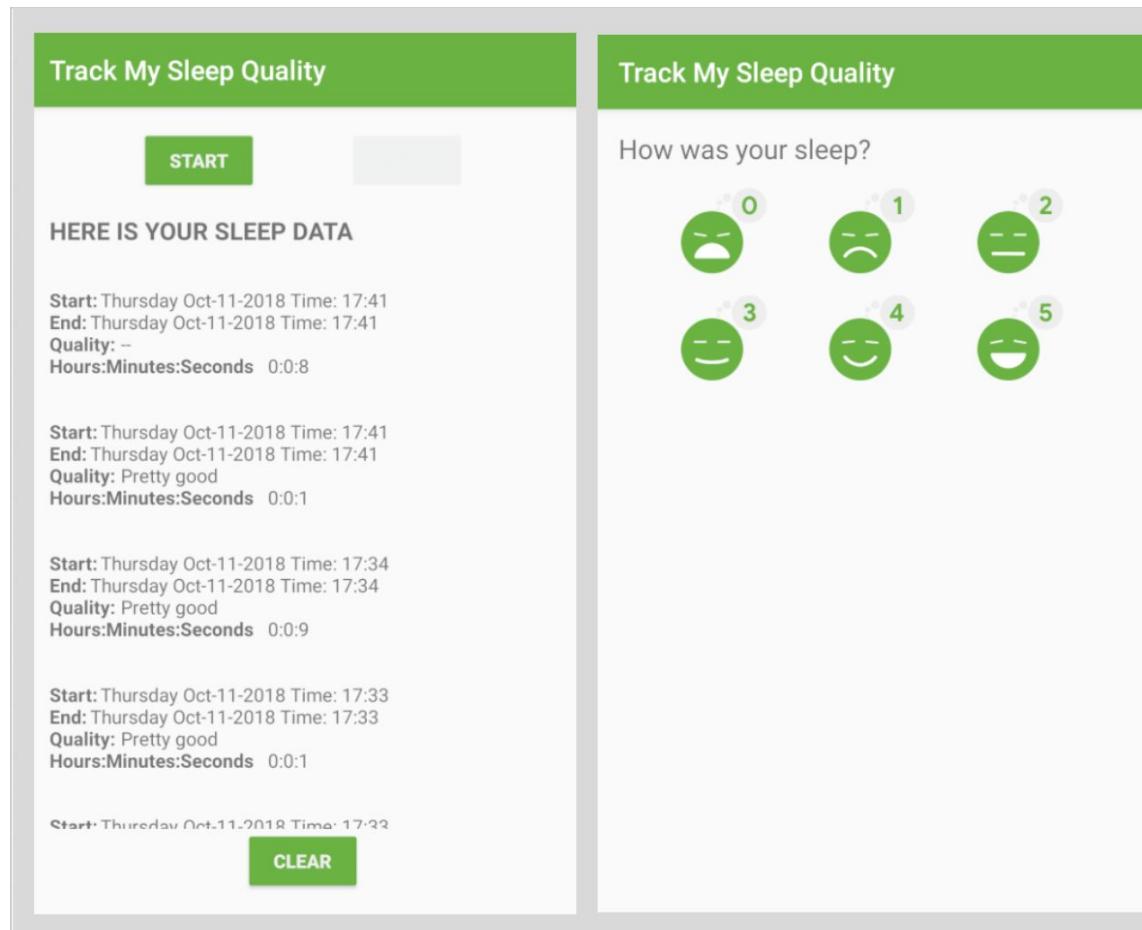
## Kotlin Features - Data classes

- Clases para almacenar datos. Ejemplo:

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

# Codelab #19: Create a Room Database

<https://bit.ly/3nWJBfh>



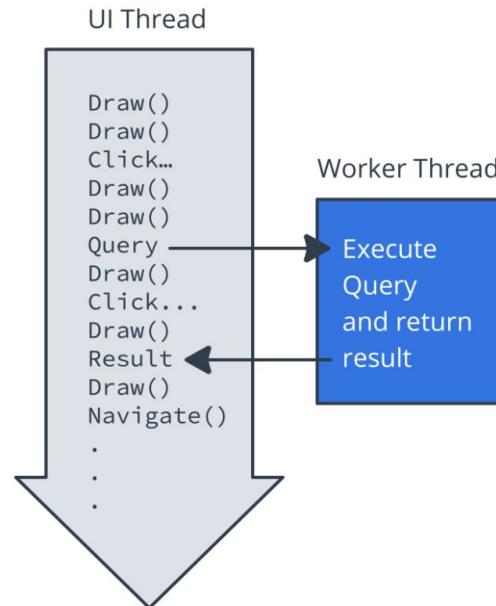
Starting App

# Testing

<https://developer.android.com/training/testing>

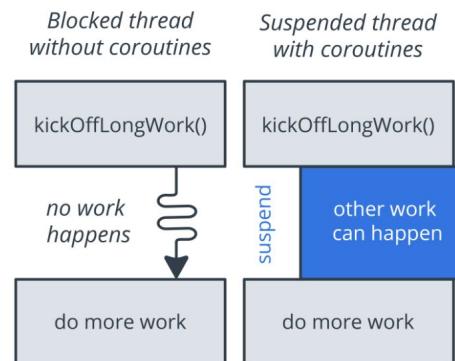
# Coroutines (I)

- Es una forma para ejecutar tareas largas eficiente y elegantemente.
- Permite convertir código con callbacks en código secuencial.
- Callbacks y coroutines hacen la misma cosa: esperan a que un resultado esté disponible después una tarea larga y continua la ejecución.



## Coroutines (II)

- Tiene las siguientes propiedades:
- ◆ Son asíncronos.
  - ◆ Son no bloqueantes.
  - ◆ Usa funciones suspendidas para hacer secuencial a un código asíncrono. Se utiliza la palabra reservada: `suspend`



# Componentes de un Coroutine

- Para utilizar coroutines en Kotlin se necesitan 3 cosas:
  - ◆ **Un job:** cada coroutine tiene un job que puede utilizarse para cancelar la corutina.
  - ◆ **Un dispatcher:** envía a que se ejecuten las corutinas en varios hilos.
  - ◆ **Un scope:** define el contexto en el que la corutina se ejecuta.  
Combina la información entre el job y el dispatcher de la corutina.

# Kotlin Coroutines + Architecture Components

- **Coroutine Scope:** hace seguimiento de todas las coroutines y administra cuando las corutinas deberían ejecutarse.
- **Componentes de Arquitectura:** define los siguientes scopes que podemos usar en nuestras aplicaciones:
  - ◆ **ViewModelScope:** definido para cada ViewModel.
  - ◆ **LifecycleScope**
  - ◆ **liveData**
- Dispatcher.IO: Para ejecutar operaciones de base de datos en background.

# Codelab #20: Coroutines and Room

<https://bit.ly/3791JvB>

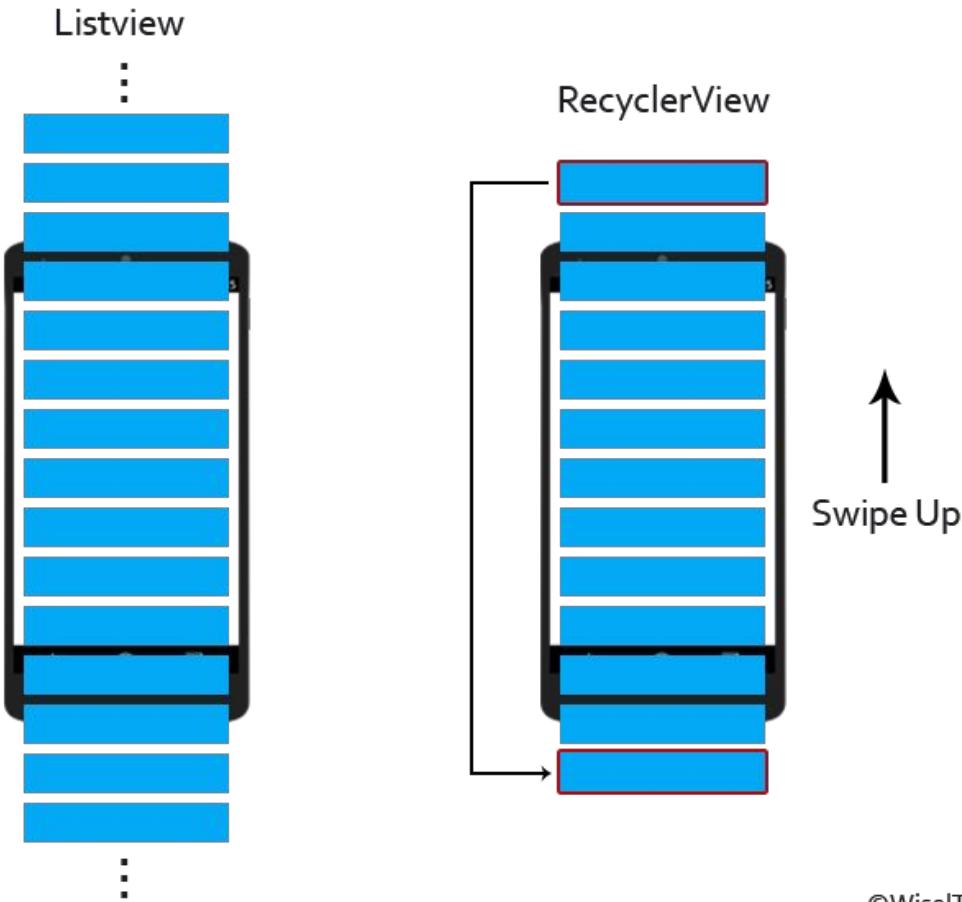
[Starting App](#)

# Codelab #21: Use LiveData to control button states

<https://bit.ly/3o37sdh>

[Starting App](#)

# RecyclerView (I)



©WiseTeach

<https://developer.android.com/guide/topics/ui/layout/recyclerview>

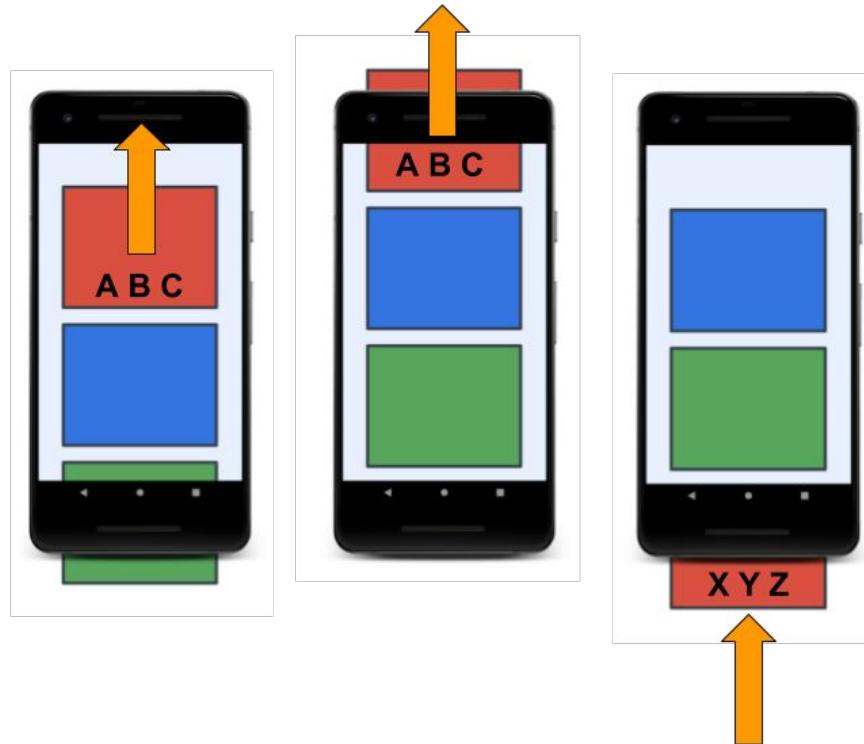
## RecyclerView (II)

- Es un widget avanzado para mostrar una lista o grilla de datos.



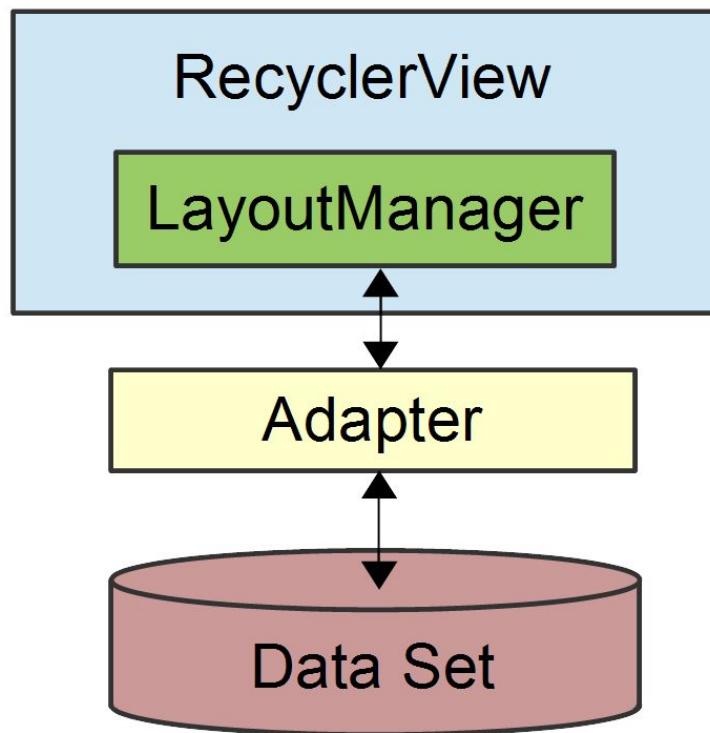
## RecyclerView (III)

- Si tenemos 1000 items para mostrar pero solo son visibles 10 se crean views para mostrar solo 10 y cuando el usuario se desliza por los items se van reutilizando las vistas.



# RecyclerView y el patrón Adapter

- RecyclerView utiliza un adapter para transformar datos en algo que pueda mostrar sin importar cómo se almacenan o procesan estos datos.



# Implementando RecyclerView

- Para mostrar datos en un RecyclerView se necesita:
  - a. Datos para mostrar.
  - b. Un contenedor RecyclerView definido en el layout.
  - c. Un layout para un item de los datos.
  - d. Un Layout Manager.
  - e. Un ViewHolder.
  - f. Un Adapter.

# Codelab #22: RecyclerView Fundamentals

<https://bit.ly/3nZSHrr>

[Starting App](#)

# **Codelab #23: Diff Util Data binding with RecyclerView**

<https://bit.ly/3mad4Sq>

[Starting App](#)

# Codelab #24: GridLayout with RecyclerView

<https://bit.ly/3mf6s52>

[Starting App](#)

# Codelab #25: Interacting with RecyclerView Items

<https://bit.ly/2JgZaPl>

[Starting App](#)

# Codelab #26: Headers in RecyclerView

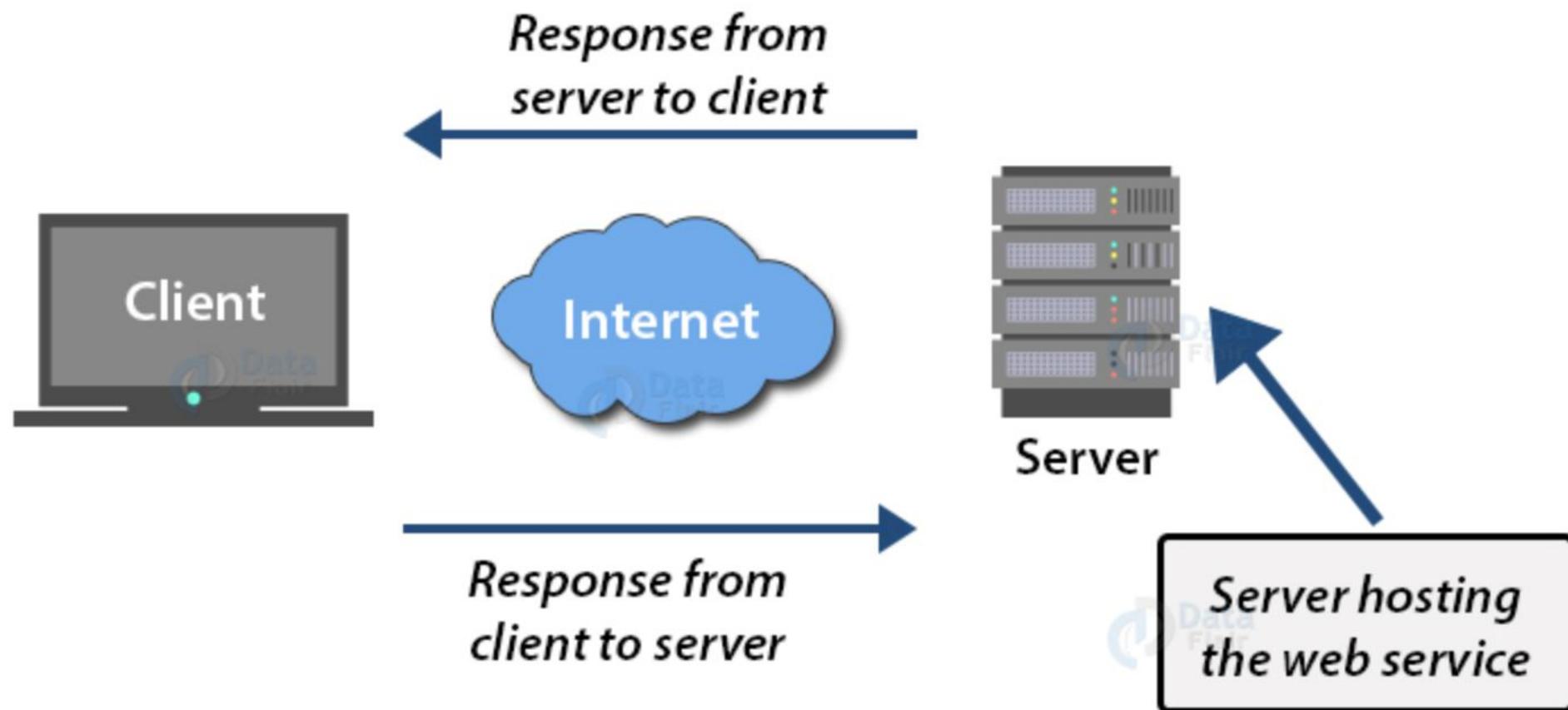
<https://bit.ly/3753oC8>

[Starting App](#)

# **Getting data from Internet**

# REST Web Services

Web Service es un método de comunicación entre dos dispositivos sobre Internet que permite hacer requests y obtener datos.



# Retrofit

- Es una librería para Java y Kotlin que permite hacer requests a Web Services.
- Convierte el HTTP API en una Interface Kotlin

```
interface Mars ApiService {  
    @GET("value: \"realestate\")  
    suspend fun getProperties(): List<MarsProperty>  
}
```

- Link: <https://square.github.io/retrofit/>

# JSON

- La respuesta de un web service suele estar en formato JSON (Javascript Object Notation), un formato común para mostrar datos estructurados.

```
[  
  {  
    "price": 450000,  
    "id": "424905",  
    "type": "buy",  
    "img_src": "http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631300503690E01\_DXXX.jpg"  
  },  
  {  
    "price": 8000000,  
    "id": "424906",  
    "type": "rent",  
    "img_src": "http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631300305227E03\_DXXX.jpg"  
  },  
  {  
    "price": 11000000,  
    "id": "424907",  
    "type": "rent",  
    "img_src": "http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000MR0044631290503689E01\_DXXX.jpg"  
  },  
]
```

- Moshi es una librería para convertir la respuesta JSON en objetos Kotlin: <https://github.com/square/moshi>

# Permisos

- Permiten al usuario decidir hacer uso de ciertos recursos del dispositivo.
- Para hacer uso de Internet por ejemplo se necesita el permiso:

```
<uses-permission android:name="android.permission.INTERNET" />
```

- Mas sobre permisos:

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

# Codelab #27: Getting Data from the Internet

<https://bit.ly/2KQaYtf>



[Starting App](#)

# Glide

- Es una librería que nos permite cargar y mostrar imágenes desde Internet.
- Glide necesita dos cosas:
  - ◆ La URL de la imagen.
  - ◆ Un widget ImageView para mostrar la imagen.
- <https://github.com/bumptech/glide>

# Codelab #28: Loading images from the Internet

<https://bit.ly/2KQaYtf>



[Starting App](https://bit.ly/2KQaYtf)

# Codelab #29: Filtering images and detail views

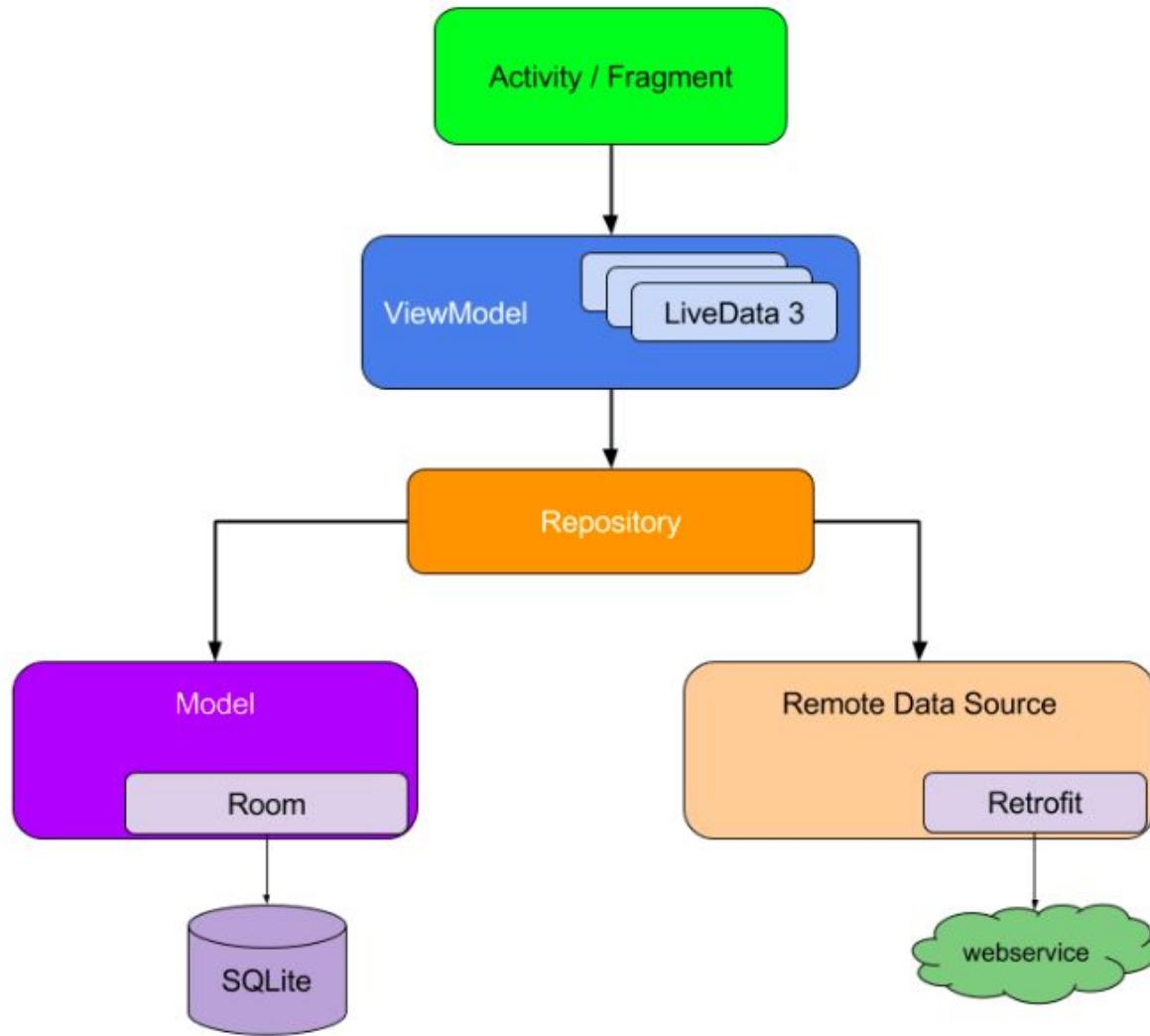
<https://bit.ly/2KQaYtf>



## **El patrón Repository**

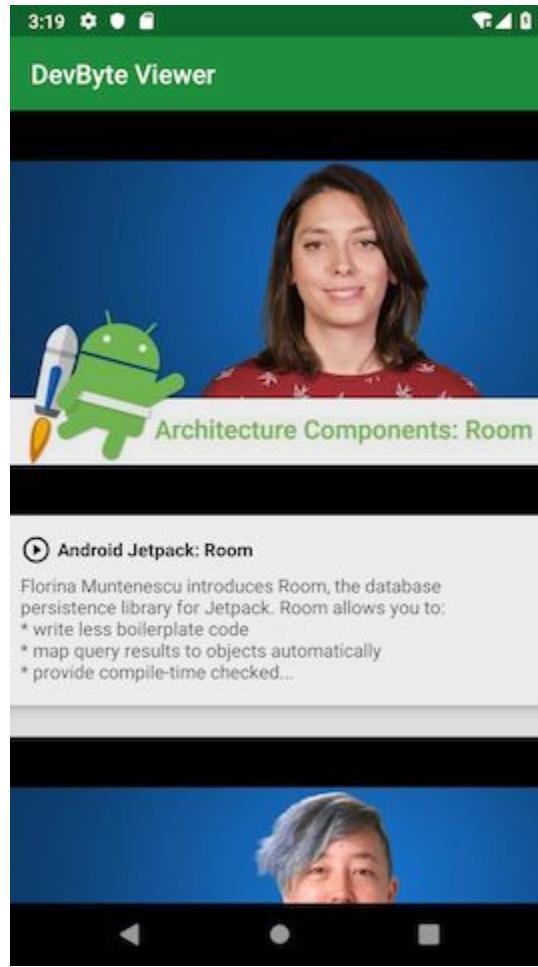
- Es un patrón de diseño que aisla las fuentes de datos y provee un API de acceso a los datos para el resto de la aplicación.
- Repository intermedia entre las fuentes de datos (modelos persistentes, servicios web y caches) y el resto de la aplicación.

# El patrón Repository



# Codelab #30: Repository

<https://bit.ly/2KQaYtf>



[Starting App](#)

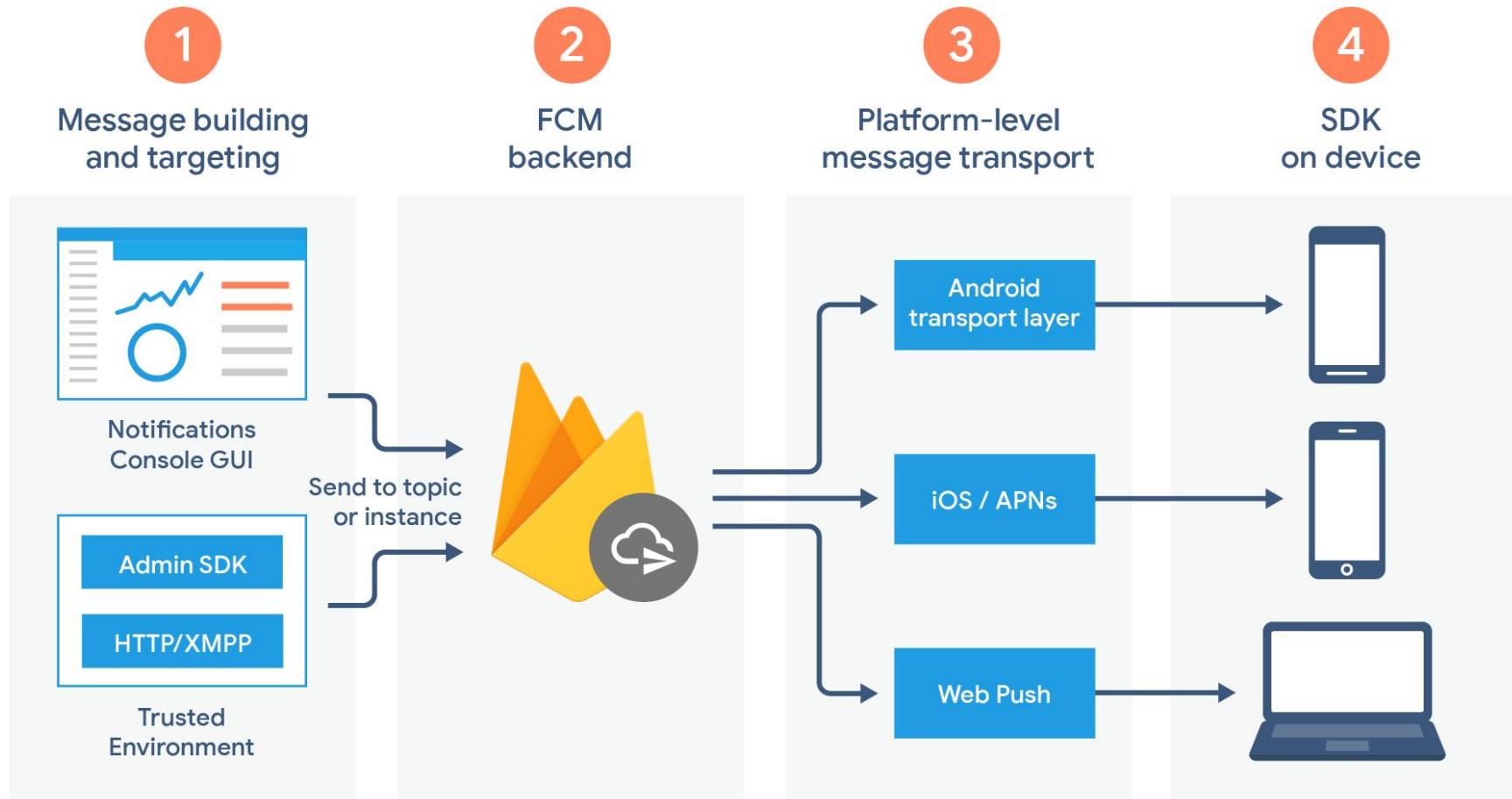
# Notifications

<https://developer.android.com/guide/topics/ui/notifiers/notifications>

# Firebase Cloud Messaging

<https://firebase.google.com/docs/cloud-messaging>

# Cómo funciona FCM?



# Adicionando Firebase al proyecto Android

1. En Android Studio ir a **Tools > Firebase**
2. Ir a la opción Cloud Messaging.
  - a. Seleccionar **Set up Firebase Cloud Messaging.**
3. En **Set up Firebase Cloud Messaging:**
  - a. Connect your app to Firebase
  - b. Add FCM to your app
  - c. Handle messages
  - d. Access the device registration token

# Componentes de una App Android

- Son los bloques esenciales de construcción de Apps Android.
- Cada componente es un punto de entrada que el sistema o el usuario puede utilizar para utilizar la app.
- Son 4, cada uno con diferente propósito:
  - ◆ Activities
  - ◆ Services
  - ◆ Broadcast Receivers
  - ◆ Content Providers

## **II) Services**

<https://developer.android.com/guide/components/services.html>

# **Services**

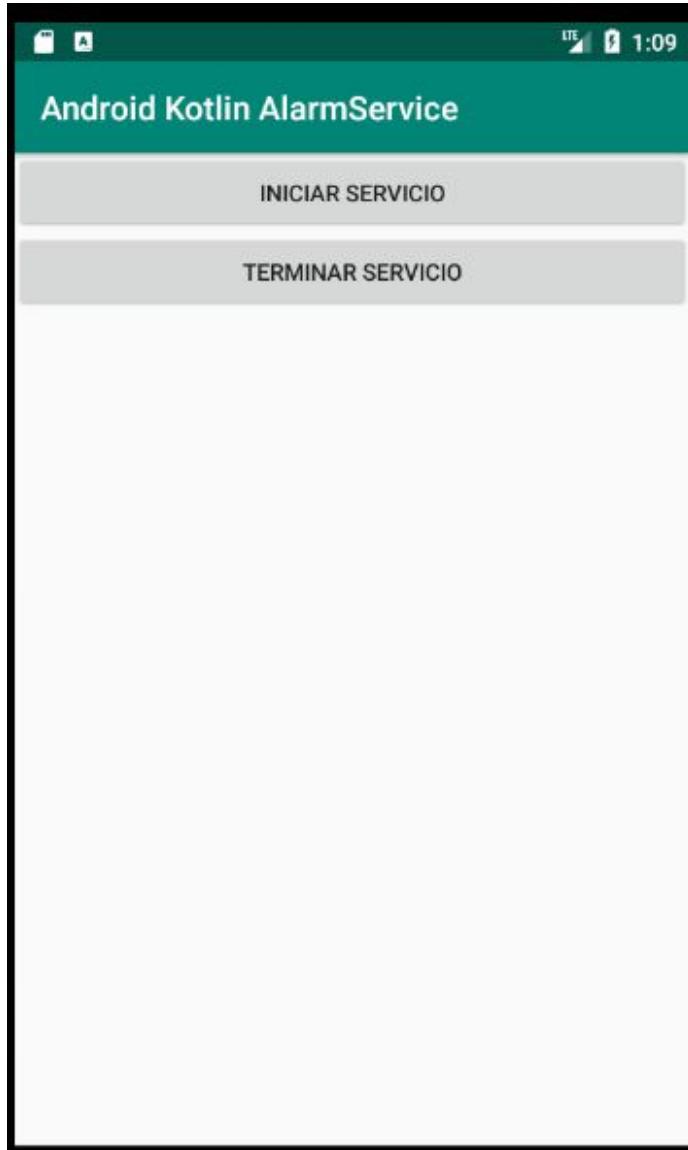
- Operaciones en Background
- No tiene UI

# Services

→ Algunos ejemplos:

- ◆ Network transactions
- ◆ Play music
- ◆ Perform file I/O operations
- ◆ Interact with content providers
- ◆ Background operations

# Ejemplo: Servicio de Alarma



# Algo de código

```
package com.teknhe.androidkotlinalarmservice

import android.app.Service
import android.content.Intent
import android.media.Ringtone
import android.media.RingtoneManager
import android.os.IBinder

class AlarmService : Service() {
    private lateinit var ringtone: Ringtone

    override fun onCreate() {
        val notification =
            RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM)
        ringtone = RingtoneManager.getRingtone(this, notification)
        super.onCreate()
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        playAlarm()
        return super.onStartCommand(intent, flags, startId)
    }

    private fun playAlarm() {
        println("playAlarm()")

        if(!ringtone.isPlaying()) {
            ringtone.play()
        }
    }

    override fun onDestroy() {
        println("onDestroy()")
        ringtone.stop()
        super.onDestroy()
    }

    override fun onBind(intent: Intent?): IBinder? = null
}
```

```
package com.teknhe.androidkotlinalarmservice

import android.content.Intent
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val iniciar = findViewById<Button>(R.id.iniciar)
        val terminar = findViewById<Button>(R.id.terminar)
        val intent = Intent(applicationContext,
            AlarmService::class.java)

        iniciar.setOnClickListener {
            view -> startService(intent)
        }

        terminar.setOnClickListener {
            view -> stopService(intent)
        }
    }
}

<service android:name=".AlarmService"/>
```

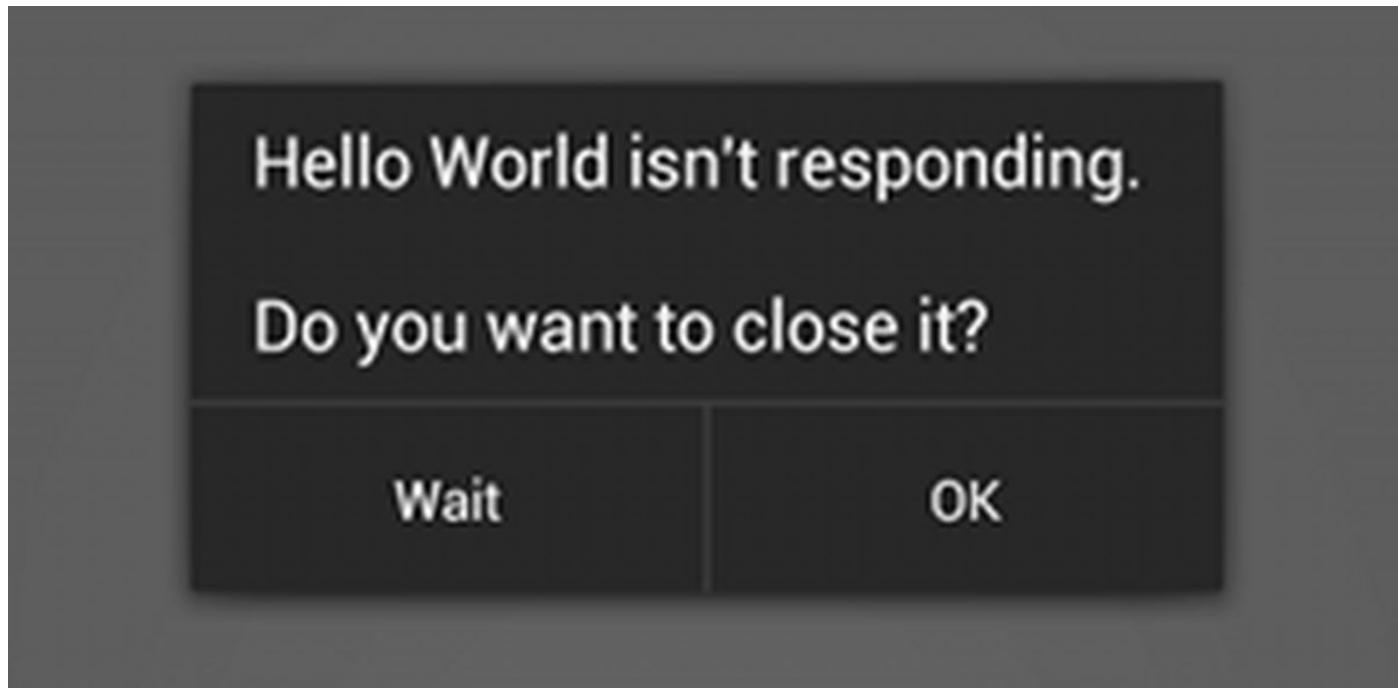
# Background Tasks

<https://developer.android.com/training/best-background>

## Procesamiento Background en Android

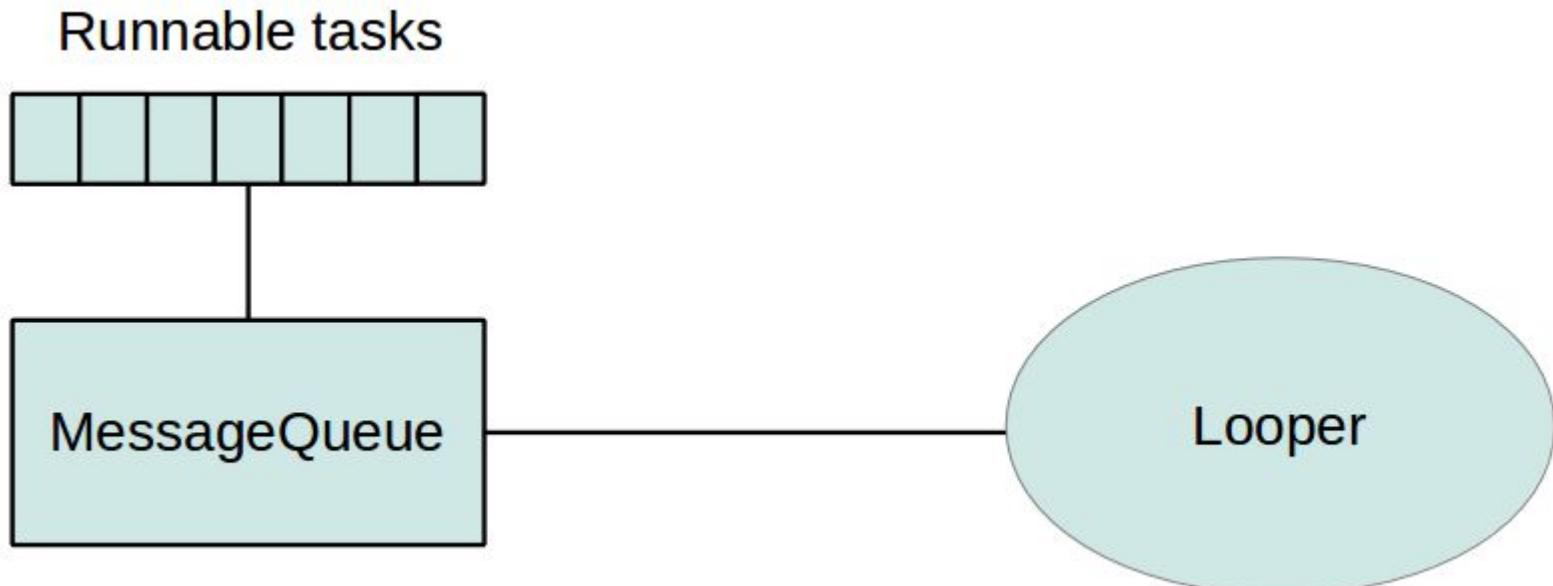
- Por defecto las aplicaciones corren en el hilo principal.
- Cada instrucción se ejecuta en secuencia.
- Si se ejecuta una operación de larga duración la aplicación se bloquea hasta ésta haya finalizado.

## Por qué usar concurrencia?

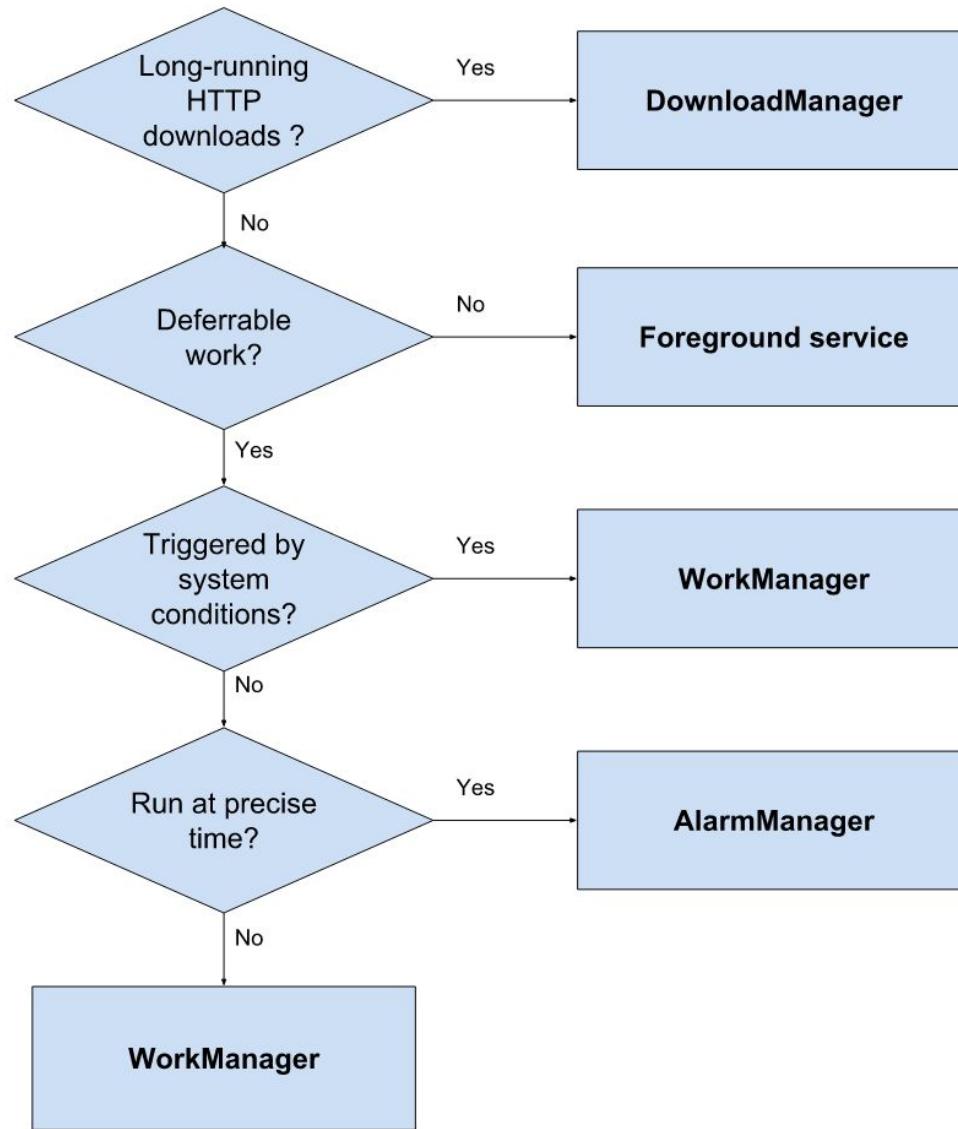


## **El hilo principal: The main thread**

- Android recolecta todos los eventos de entrada en una cola desde el hilo principal y los procesa con una instancia de la clase Looper.



# Guia al Procesamiento en Background



## **III) Broadcasts**

<https://developer.android.com/guide/components/broadcasts>

## **IV) Content Providers**

<https://developer.android.com/guide/topics/providers/content-providers>

# Activando Componentes

→ Intents

- ◆ Activity
- ◆ Services
- ◆ Broadcast receivers

→ Content Resolver

- ◆ Content Provider

# Activando un Activity

[ ... ]

```
val intent = Intent(applicationContext,  
    DisplayMessageActivity::class.java)  
  
startActivity(intent)
```

[ ... ]

# Activando un Service

```
[ ... ]
```

```
val intent = Intent(applicationContext,  
    AlarmService::class.java)  
  
startService(intent)
```

```
[ ... ]
```

## Activando un Broadcast Receiver

```
[ ... ]  
  
val intent = Intent()  
  
intent.action = "com.example.Broadcast"  
  
intent.putExtra("MyData", 1000)  
  
sendBroadcast(intent)  
  
[ ... ]
```

# Activando un Content Resolver

```
[ ... ]
```

```
val cursor = contentResolver.query(  
    Uri.parse(queryUri), projection, selectionClause,  
    selectionArgs, sortOrder)
```

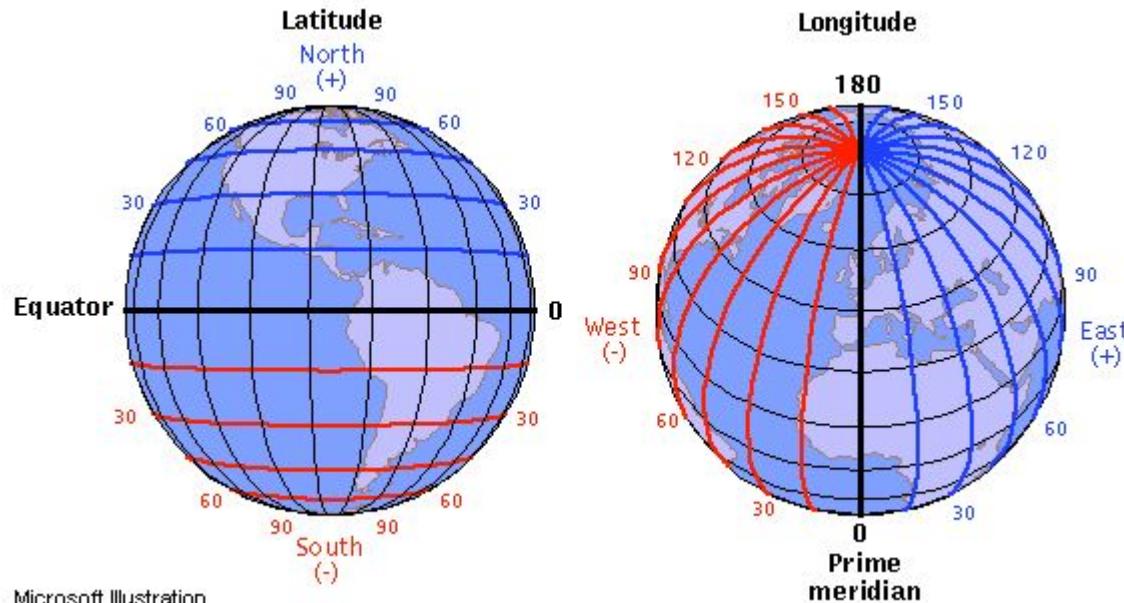
```
[ ... ]
```

# User Location

<https://developer.android.com/training/location>

# GPS

- Global Positioning System (Sistema de Posicionamiento Global)
- Sistema de navegación basada en Satélites a 20K msnm.
- Los satélites mandan datos hacia la tierra a GPS Receivers como los smartphones permitiendo calcular su posición sobre la tierra.
- Las coordenadas GPS dependen de los valores en latitud y longitud.



# Google Play Store

<https://developer.android.com/distribute>

## Google Play Store

- Pasos para poder publicar aplicaciones en la tienda oficial:
  - a. Registrarte en este enlace  
<https://play.google.com/apps/publish/signup/>
  - b. Aceptar el acuerdo de distribución para desarrolladores, lo puedes consultar aquí:  
<https://play.google.com/about/developer-distribution-agreement.html>
  - c. Pagar la única tarifa de registro de \$25 USD al día de hoy.
  - d. Completar los datos de la cuenta.
- <https://android-developers.googleblog.com/2011/06/things-that-can-not-change.html>

# **Bibliografía**

# Bibliografía

- <https://developer.android.com/>
- <https://kotlinlang.org>

## Material recomendado

- <https://developer.android.com/courses/basic-android-kotlin-training/overview>
- <https://codelabs.developers.google.com/?cat=android>