PROGRAMACIÓN DE DISPOSITIVOS MÓVILES

ESCUELA PLATÓN – Abril 2026 |JCMF|

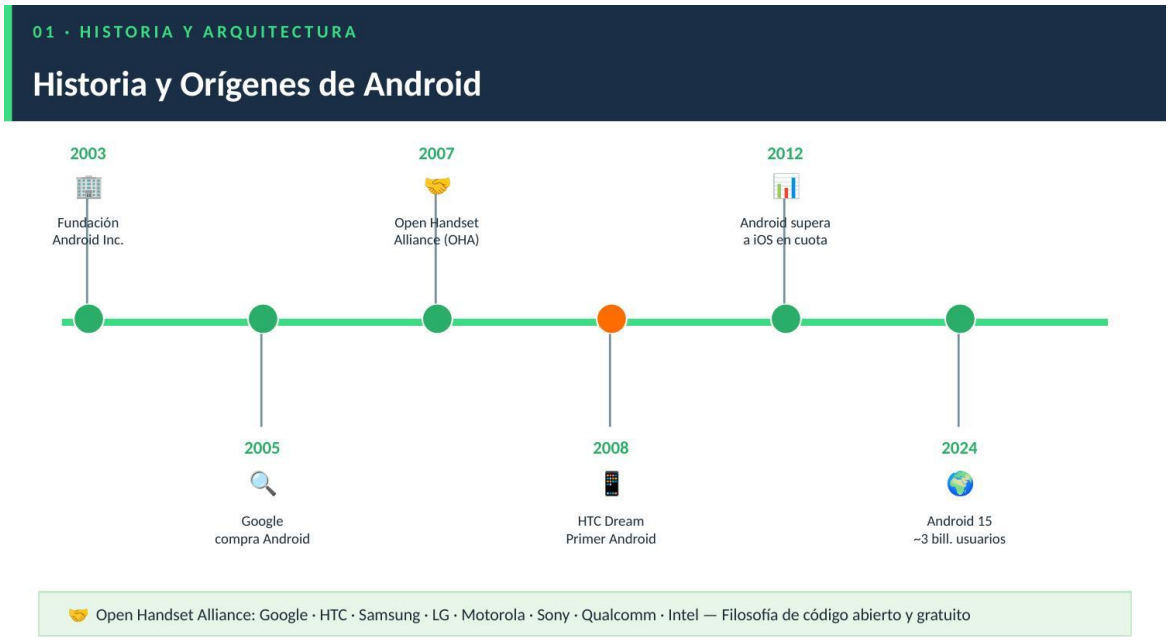
Tema 2

Introducción a Android

> *Estos apuntes cubren los fundamentos del sistema operativo Android: su historia y arquitectura, el entorno de desarrollo con Android Studio, los componentes principales de una app, la construcción de interfaces de usuario y el sistema de gestión de recursos.*

Contenidos

1. Introducción a Android



1.1. Historia y Orígenes

Fundación de Android Inc. (2003)

- **Fundadores:** Andy Rubin, Rich Miner, Nick Sears y Chris White
- **Objetivo inicial:** Sistema operativo para cámaras digitales. Pivotó rápidamente al mercado de smartphones.

Adquisición por Google:

- **2005:** Google adquiere Android Inc. por ~50 millones de dólares
- **2007:** Anuncio del Open Handset Alliance (OHA)
- **2008:** Lanzamiento del primer dispositivo Android — HTC Dream / T-Mobile G1

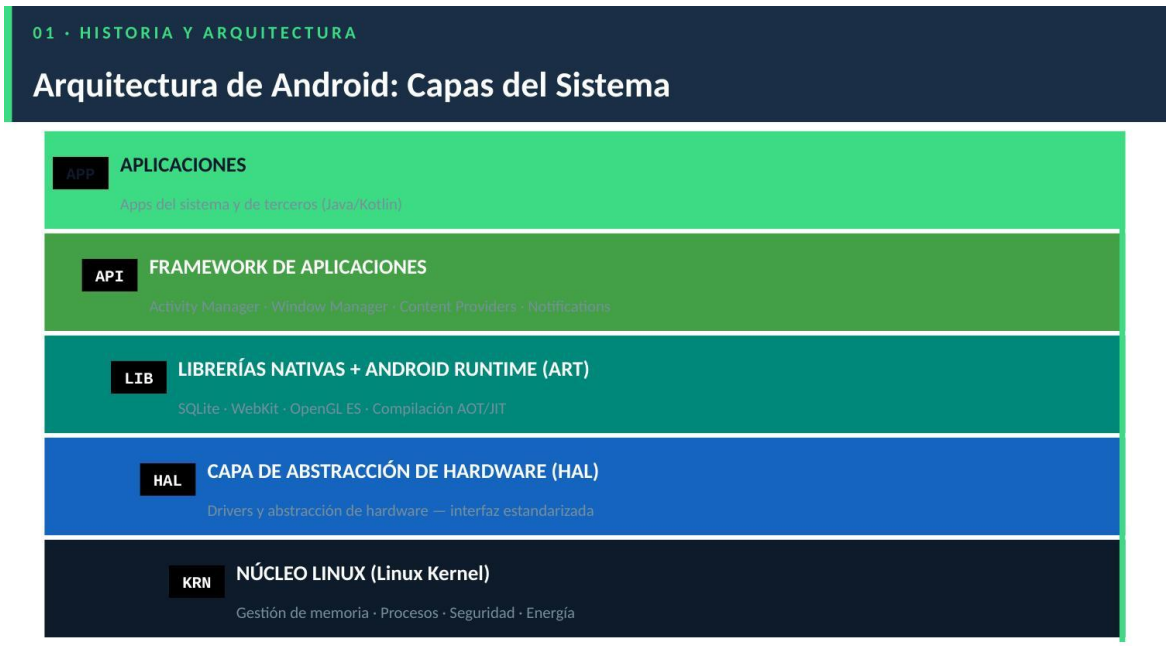
Open Handset Alliance (OHA):

Consorcio de empresas tecnológicas para desarrollar estándares abiertos para móviles, con la filosofía de un sistema completamente abierto y gratuito.

- **Miembros fundadores:** Google, HTC, Samsung, LG, Motorola, Sony, Qualcomm, Intel y otras 80+ empresas

> 💡 *Dato clave: En 2024, Android supera los 3.000 millones de usuarios activos en todo el mundo, con una cuota de mercado del ~70% en smartphones.*

1.2. Arquitectura de Android



Android utiliza una arquitectura en capas donde cada nivel provee servicios a la capa superior:

Capa	Componente	Descripción
1 (Superior)	Aplicaciones	Apps del sistema y de terceros (Java/Kotlin)
2	Framework de Aplicaciones	Activity Manager, Window Manager, Content Providers, Notifications
3	Librerías Nativas + ART	SQLite, WebKit, OpenGL ES — Android Runtime (AOT+JIT)
4	HAL	Capa de Abstracción de Hardware — drivers estandarizados
5 (Inferior)	Linux Kernel	Gestión de memoria, procesos, seguridad y energía

Framework de Aplicaciones

- **Activity Manager:** Gestiona el ciclo de vida de las apps
- **Window Manager:** Gestiona ventanas e interfaz de usuario
- **Content Providers:** Comparte datos entre aplicaciones
- **View System:** Construcción de interfaces de usuario
- **Notification Manager:** Sistema de notificaciones al usuario
- **Package Manager:** Información sobre las apps instaladas

Android Runtime (ART)

ART reemplazó a Dalvik en Android 5.0 (Lollipop). Es el entorno de ejecución de todas las apps Android.

- **Bytecode DEX:** Las apps se compilan a formato Dalvik Executable (.dex)
- **Compilación AOT:** Ahead-Of-Time — compila al instalar la app, mejora el arranque
- **Compilación JIT:** Just-In-Time — optimiza en tiempo de ejecución según el uso

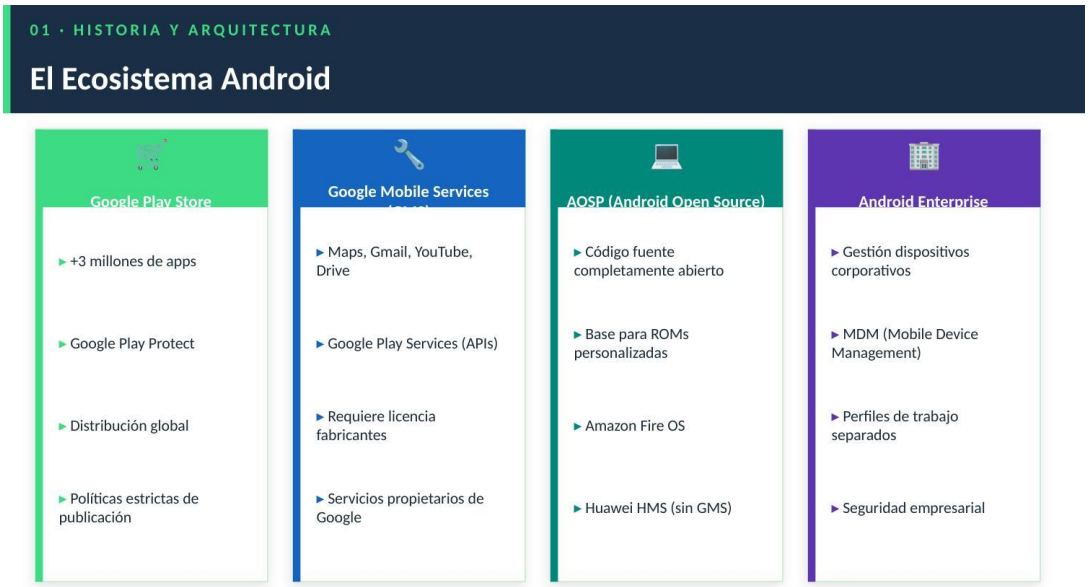
- **Ventaja:** Mejor rendimiento y eficiencia energética que Dalvik

1.3. Versiones de Android

Versión	API	Nombre	Año	Características principales
2.3	9-10	Gingerbread	2010	NFC, mejoras UI
4.0	14-15	Ice Cream Sandwich	2011	Face Unlock, unificó phone/tablet
4.1-4.3	16-18	Jelly Bean	2012-13	Google Now, Project Butter
5.0-5.1	21-22	Lollipop	2014-15	Material Design, ART, 64-bit
6.0	23	Marshmallow	2015	Permisos en tiempo de ejecución, USB-C
7.0-7.1	24-25	Nougat	2016	Multitarea, Vulkan API
8.0-8.1	26-27	Oreo	2017	Picture-in-picture, Project Treble
10	29	(sin nombre)	2019	5G nativo, modo oscuro oficial
12	31-32	Material You	2021	Personalización dinámica, privacidad mejorada
14	34	(sin nombre)	2023	Personalización, rendimiento mejorado
15	35	(sin nombre)	2024	IA integrada, audio espacial

> *Distribución actual (2024): Android 14 ~30% · Android 13 ~25% · Android 12 ~18% · Android 11 o inferior ~27%. Esta fragmentación es uno de los principales retos para los desarrolladores.*

1.4. Ecosistema Android



Google Play Store

- Más de 3 millones de aplicaciones disponibles
- Distribución global (con restricciones en algunos países)
- Google Play Protect: análisis de seguridad automatizado
- Políticas de publicación estrictas y revisión de contenido

Google Mobile Services (GMS)

- Conjunto de servicios propietarios de Google: Maps, Gmail, YouTube, Drive
- Google Play Services: APIs fundamentales para apps
- Los fabricantes deben obtener licencia de Google para incluirlo

Android Open Source Project (AOSP)

- El código fuente de Android es completamente abierto (licencia Apache 2.0)
- Base para ROMs personalizadas (LineageOS, GrapheneOS...)
- **Alternativas sin GMS:** Amazon Fire OS, Huawei HMS (tras sanciones comerciales)

Android Enterprise

- Gestión de dispositivos corporativos (MDM — Mobile Device Management)
- Perfiles de trabajo separados del uso personal
- Políticas de seguridad aplicables remotamente

2. Entorno de Desarrollo

2.1. Android Studio

Android Studio es el IDE (Integrated Development Environment) oficial para el desarrollo de apps Android. Está basado en IntelliJ IDEA de JetBrains y ofrece las mejores herramientas para construir, depurar y publicar aplicaciones.

Requisitos del Sistema

Componente	Mínimo	Recomendado
Sistema Operativo	Windows 8+, macOS 10.14+, Linux	Windows 10/11, macOS 12+, Linux 64-bit
RAM	4 GB	16 GB o más
Espacio en disco	2 GB	8 GB+ (SSD recomendado)
Resolución	1280×800	1920×1080 o superior

Pasos de Instalación

1. **Descargar Android Studio:** Desde https://developer.android.com/studio — siempre la versión estable más reciente
2. **Ejecutar el instalador:** .exe en Windows · Arrastrar a Applications en macOS · studio.sh en Linux
3. **Configuración inicial:** Seleccionar tipo Standard y elegir tema (Light / Darcula)
4. **SDK Setup:** Descargar el Android SDK, seleccionar componentes y aceptar licencias
5. **Finalizar:** Descargar componentes adicionales. El IDE queda listo para usar.

2.2. Configuración del SDK

Android SDK Manager es la herramienta integrada en Android Studio para gestionar todos los componentes del SDK:

```
SDK Platform     → Versiones de Android disponibles para desarrollo
SDK Tools        → Herramientas de desarrollo (emulador, etc.)
SDK Build-Tools  → Compiladores para cada API Level (aapt, dx, zipalign...)
SDK Platform-Tools → adb, fastboot
```

Componente	Descripción
Android SDK Platform	Sistema Android para cada versión objetivo
SDK Build-Tools	Herramientas de compilación: aapt, dx, etc.
SDK Platform-Tools	adb (Android Debug Bridge) y fastboot
Android Support Repository	Librerías de compatibilidad (AndroidX)
Google Repository	Google Play Services y otras APIs de Google

2.3. Estructura de un Proyecto Android

Todo proyecto Android sigue una estructura estandarizada de directorios. Conocerla es fundamental para trabajar de forma eficiente:

```
MiApp/
├── app/
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/com.ejemplo.miapp/
│   │   │   │   └── MainActivity.kt       ← Código Kotlin
│   │   │   ├── res/
│   │   │   │   ├── layout/
│   │   │   │   │   └── activity_main.xml ← Layout de la pantalla
│   │   │   │   ├── values/
│   │   │   │   │   ├── strings.xml        ← Textos
│   │   │   │   │   ├── colors.xml         ← Colores
│   │   │   │   │   └── themes.xml         ← Temas
│   │   │   │   └── drawable/              ← Imágenes e iconos
│   │   │   └── AndroidManifest.xml        ← Configuración global
│   ├── build.gradle.kts                   ← Dependencias
└── settings.gradle.kts
```

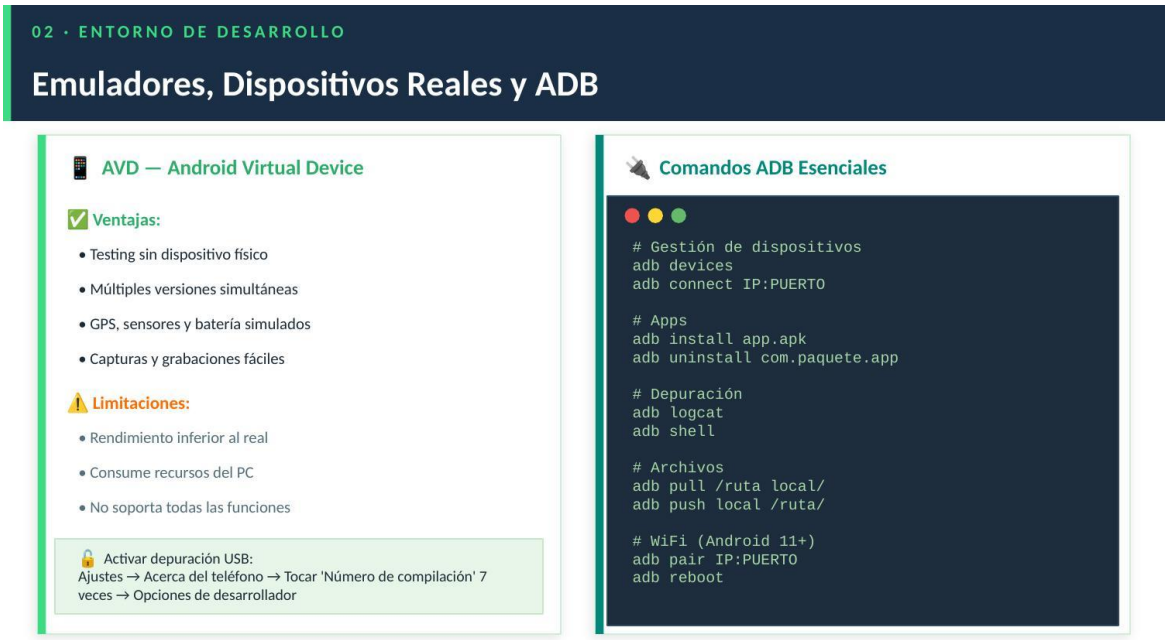
AndroidManifest.xml

El fichero AndroidManifest.xml es el registro central de la aplicación. Define permisos, componentes registrados y configuración global:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-permission android:name="android.permission.INTERNET"/>
```

```
  <uses-permission android:name="android.permission.CAMERA"/>
  <application
      android:icon="@mipmap/ic_launcher"
      android:theme="@style/Theme.MiApp">
    <activity android:name=".MainActivity" android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

2.4. Emuladores, Dispositivos y ADB



Android Virtual Device (AVD)

El AVD Manager permite crear emuladores de dispositivos Android con diferentes configuraciones:

Pasos para crear un AVD:

6. Tools → Device Manager → Create Device
7. Seleccionar hardware (Pixel 7, Galaxy Nexus...)
8. Seleccionar imagen del sistema (versión Android / API Level)
9. Configurar opciones avanzadas: RAM, almacenamiento, GPU
10. Finalizar y lanzar el emulador

Ventajas del emulador:

- Testing en múltiples versiones de Android sin dispositivos físicos
- Capturas de pantalla y grabaciones de vídeo fáciles
- Simulación de GPS, sensores y estado de batería

Limitaciones:

- Rendimiento inferior a un dispositivo real
- No soporta todas las funciones de hardware (NFC, Bluetooth limitado)
- Consume recursos importantes del ordenador

Depuración en Dispositivo Real

11. **Activar opciones de desarrollador:** Ajustes → Acerca del teléfono → Tocar 'Número de compilación' 7 veces
12. **Activar depuración USB:** Opciones de desarrollador → Depuración USB
13. **Conectar por USB:** Aceptar el permiso de depuración en el dispositivo
14. **Verificar conexión:** Ejecutar adb devices en la terminal

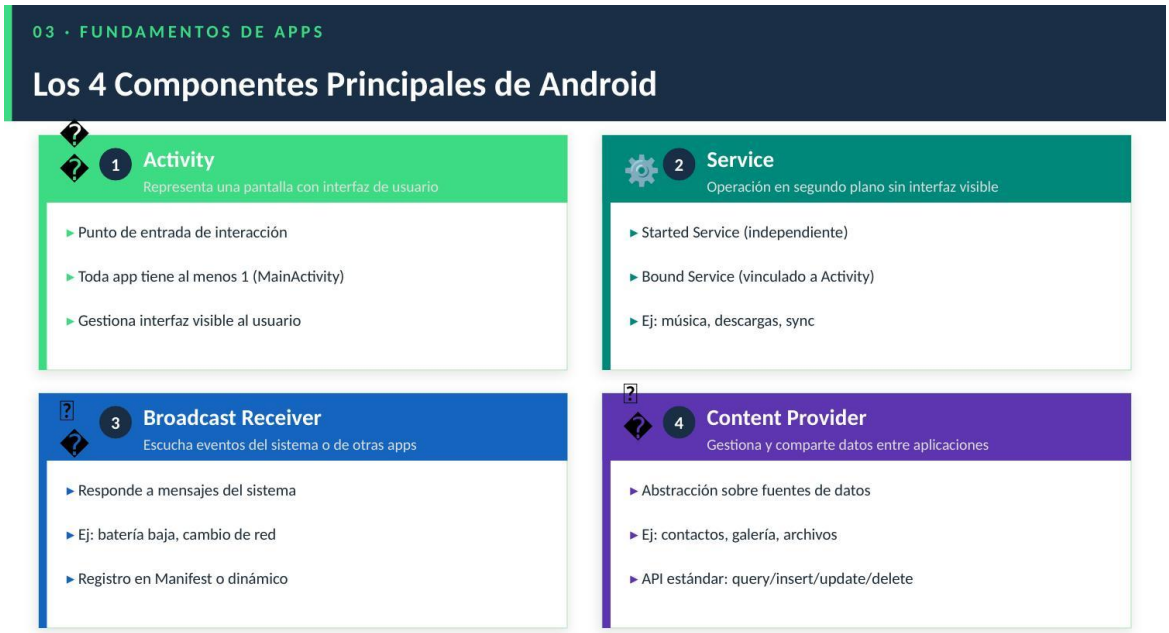
Comandos ADB Esenciales

ADB (Android Debug Bridge) es la herramienta de línea de comandos para comunicarse con el dispositivo:

```
adb devices                        # Listar dispositivos conectados
adb install app.apk                # Instalar un APK
adb uninstall com.paquete.app      # Desinstalar una app
adb logcat                         # Ver logs en tiempo real
adb shell                          # Abrir shell remota en el dispositivo
adb pull /sdcard/archivo local/    # Copiar archivo del dispositivo al PC
adb push archivo.txt /sdcard/      # Copiar archivo del PC al dispositivo
adb reboot                         # Reiniciar el dispositivo

# WiFi Debugging (Android 11+)
adb pair IP:PUERTO                 # Emparejar dispositivo por WiFi
adb connect IP:PUERTO              # Conectar vía WiFi
```

3. Fundamentos de las Aplicaciones Android



3.1. Los 4 Componentes Principales

Android define cuatro tipos fundamentales de componentes que son los bloques de construcción de cualquier app:

1. Activity (Actividad)

Representa una pantalla con interfaz de usuario. Es el punto de entrada para la interacción con el usuario.

- Toda aplicación tiene al menos una Activity (normalmente MainActivity)
- Gestiona la interfaz visible al usuario en un momento dado
- Tiene su propio ciclo de vida que hay que gestionar correctamente

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

2. Service (Servicio)

Componente para ejecutar operaciones en segundo plano sin interfaz de usuario visible.

- **Started Service:** Se inicia con startService() y se ejecuta indefinidamente
- **Bound Service:** Se vincula a una Activity; se destruye cuando todos los clientes se desconectan
- **Ejemplos de uso:** Reproducción de música, descarga de archivos, sincronización de datos

```
class MyService : Service() {
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
Int {
        // Trabajo en segundo plano
        return START_STICKY  // Se reinicia si el sistema lo mata
    }
    override fun onBind(intent: Intent?): IBinder? = null
}
```

3. BroadcastReceiver (Receptor de Difusión)

Escucha y responde a eventos del sistema o de otras aplicaciones.

- Se puede registrar en el AndroidManifest.xml o dinámicamente en código
- **Ejemplos de eventos:** Batería baja, cambio de red, arranque del dispositivo, foto tomada

```
class BatteryReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (intent.action == Intent.ACTION_BATTERY_LOW) {
            // Accionar cuando la batería está baja
        }
    }
}
```

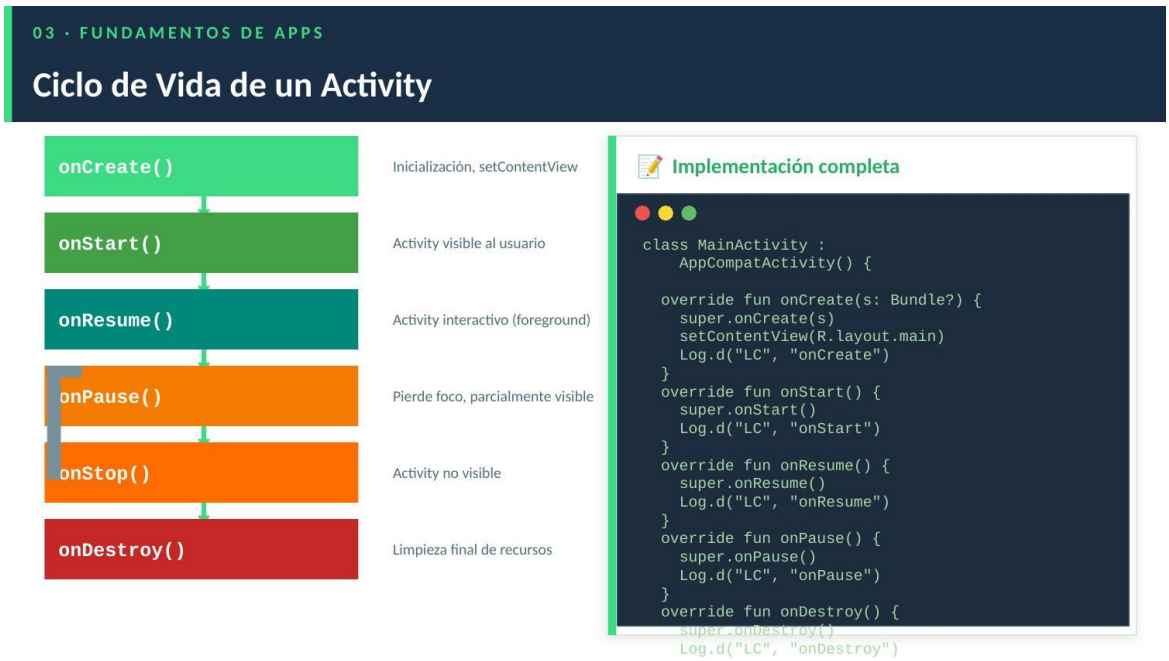
4. ContentProvider (Proveedor de Contenido)

Gestiona y comparte datos entre aplicaciones de forma estructurada.

- Proporciona una API estándar: query, insert, update, delete
- **Ejemplos:** Acceder a contactos del dispositivo, galería de fotos, datos de otras apps

```
class MiProvider : ContentProvider() {
    override fun query(uri: Uri, ...): Cursor? { ... }
    override fun insert(uri: Uri, ...): Uri? { ... }
    override fun update(...): Int { ... }
    override fun delete(...): Int { ... }
    override fun onCreate(): Boolean = true
}
```

3.2. Ciclo de Vida de un Activity



Comprender el ciclo de vida es fundamental para desarrollar apps robustas que no pierdan datos ni consuman recursos innecesariamente:

Método	Cuándo se llama	Qué hacer
onCreate()	Primera creación de la Activity	Inicialización, setContentView(), vincular vistas
onStart()	Activity se hace visible	Preparar la interfaz de usuario
onResume()	Activity pasa a primer plano	Iniciar animaciones, listeners, cámara
onPause()	Otra Activity toma el foco	Pausar animaciones, guardar estado temporal
onStop()	Activity deja de ser visible	Liberar recursos pesados
onDestroy()	Antes de destruir la Activity	Limpieza final de todos los recursos
onRestart()	Después de onStop(), antes de onStart()	Preparar para volver a ser visible

> ⚠ *Regla de oro: En onPause() y onStop() hay que liberar recursos que consuman CPU o memoria. En onCreate() y onResume() hay que volver a inicializarlos. Nunca hacer operaciones largas en onCreate().*

3.3. Intents y Navegación

Los Intents son el mecanismo de comunicación entre componentes en Android. Hay dos tipos:

Intent Explícito

Especifica exactamente el componente destino. Se usa para navegar dentro de la misma app.

```
// Navegar a otra Activity
val intent = Intent(this, SecondActivity::class.java)
// Pasar datos con extras
intent.putExtra("KEY_NAME", "Valor a pasar")
startActivity(intent)

// Recibir los datos en SecondActivity
val valor = intent.getStringExtra("KEY_NAME")
```

Intent Implícito

Declara la acción a realizar sin especificar el componente. El sistema elige la app más adecuada.

```
// Abrir URL en el navegador
val intent = Intent(Intent.ACTION_VIEW)
intent.data = Uri.parse("https://www.ejemplo.com")
startActivity(intent)

// Llamar por teléfono
val intent = Intent(Intent.ACTION_DIAL)
intent.data = Uri.parse("tel:+34600123456")
startActivity(intent)

// Compartir contenido de texto
val intent = Intent(Intent.ACTION_SEND)
intent.type = "text/plain"
intent.putExtra(Intent.EXTRA_TEXT, "Texto a compartir")
startActivity(Intent.createChooser(intent, "Compartir con"))
```

3.4. Fragmentos (Fragments)

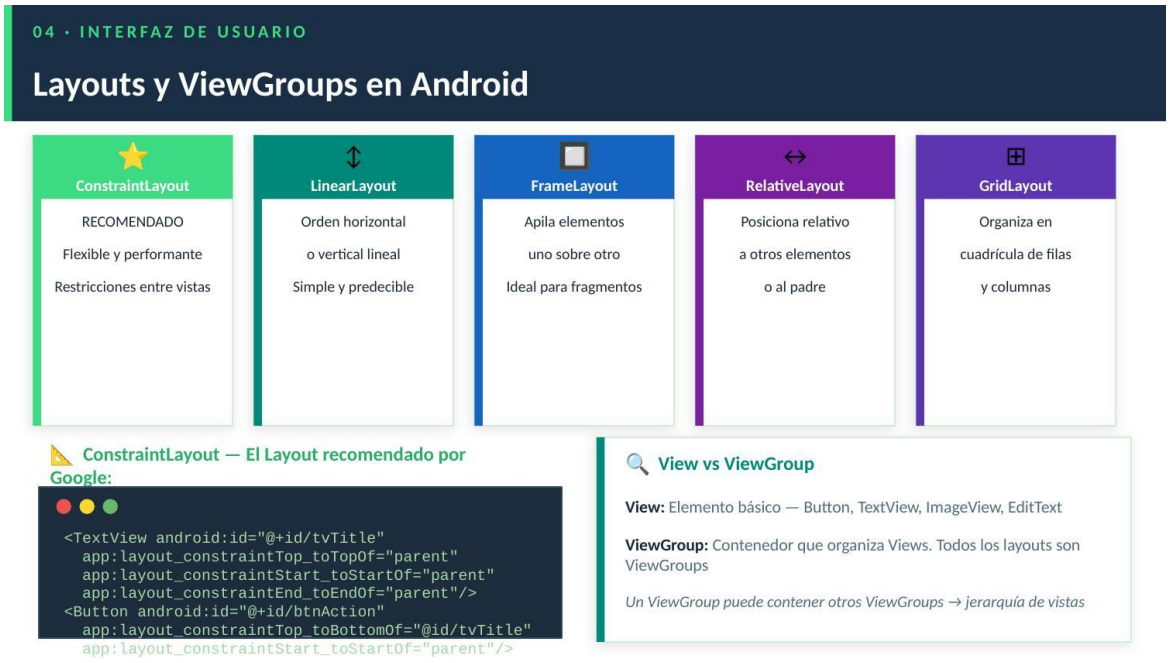
Definición: Un Fragment es una porción de UI reutilizable que vive dentro de una Activity. Permite construir interfaces modulares y adaptables a diferentes tamaños de pantalla.

- Un Fragment tiene su propio ciclo de vida, similar al de un Activity
- Se puede añadir, reemplazar o eliminar de forma dinámica
- Ideal para diseños donde tablets muestran más contenido que teléfonos

Gestión dinámica de Fragments

```
// Añadir un Fragment dinámicamente
supportFragmentManager.beginTransaction()
    .replace(R.id.fragment_container, MiFragment())
    .addToBackStack(null)  // Permite volver con el botón Back
    .commit()
```

4. Interfaz de Usuario



4.1. Layouts y ViewGroups

View vs ViewGroup: Una View es un elemento básico de UI (Button, TextView). Un ViewGroup es un contenedor que organiza Views. Todos los layouts son ViewGroups.

Tipos de Layouts

1. ConstraintLayout (Recomendado por Google)

- Layout flexible y de alto rendimiento
- Posiciona elementos mediante restricciones (constraints) entre ellos o respecto al padre
- Ideal para diseños complejos sin anidar múltiples layouts

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/tvTitulo"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="16dp"/>

    <Button
        android:id="@+id/btnAccion"
        app:layout_constraintTop_toBottomOf="@id/tvTitulo"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

2. LinearLayout

- Organiza elementos en una sola fila (horizontal) o columna (vertical)
- `android:orientation="vertical"` o `"horizontal"`

3. FrameLayout

- Apila elementos uno sobre otro
- Ideal para contener un único Fragment o superponer vistas

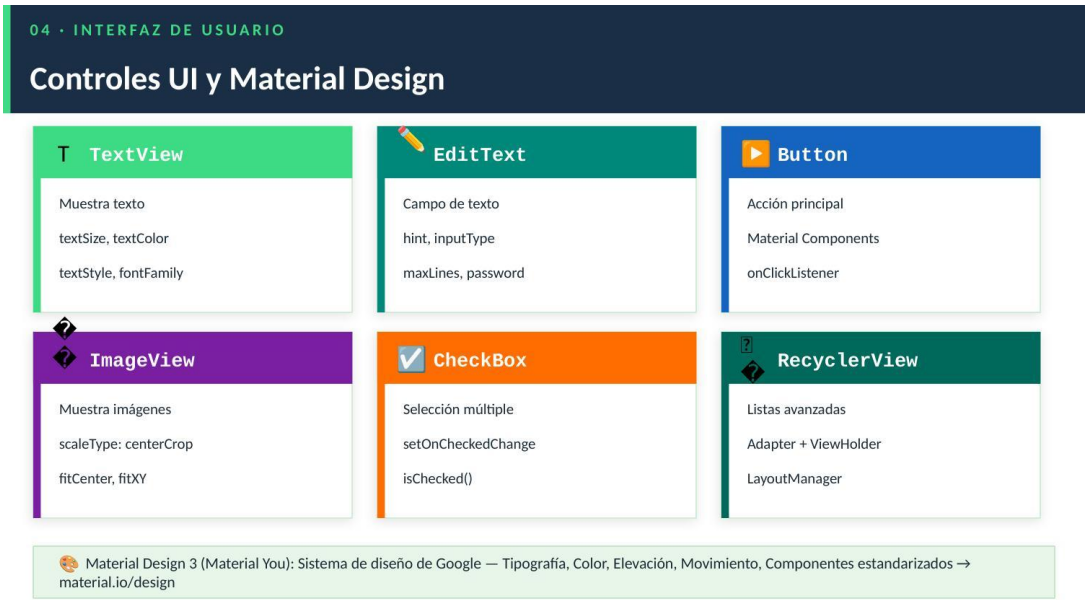
4. RelativeLayout

- Posiciona elementos relativamente a otros o al padre
- Superado por ConstraintLayout, pero aún presente en proyectos legados

5. GridLayout

- Organiza elementos en una cuadrícula de filas y columnas
- Útil para teclados, calculadoras, pantallas de categorías

4.2. Controles UI Básicos



Control	Atributos clave	Uso típico
TextView	text, textSize (sp), textColor, textStyle, fontFamily	Mostrar texto estático o dinámico
EditText	hint, inputType, maxLines, id para leer valor	Entrada de texto del usuario
Button	text, onClick, style Material	Acción principal de la pantalla
ImageView	src, scaleType (centerCrop, fitCenter)	Mostrar imágenes y iconos

Control	Atributos clave	Uso típico
CheckBox	checked, text, setOnCheckedChangeListener	Selección múltiple de opciones
RadioButton/Group	orientation, checked, getCheckedRadioButtonId()	Selección exclusiva entre opciones
Switch	checked, text, setOnCheckedChangeListener	Activar/desactivar funciones
Spinner	entries (array), setOnItemSelectedListener	Lista desplegable de opciones
RecyclerView	adapter, layoutManager, itemDecoration	Listas y cuadrículas de gran volumen

Tipos de inputType para EditText

```
text             → Texto general
textPersonName   → Nombre propio (mayúscula inicial)
textEmailAddress → Email (teclado con @)
textPassword     → Contraseña (texto oculto)
number           → Solo números
phone            → Teléfono (teclado numérico)
textMultiLine    → Texto multilínea
```

4.3. Material Design

Material Design 3 (Material You) es el sistema de diseño de Google que proporciona componentes, guías de color, tipografía y movimiento para crear interfaces coherentes y accesibles.

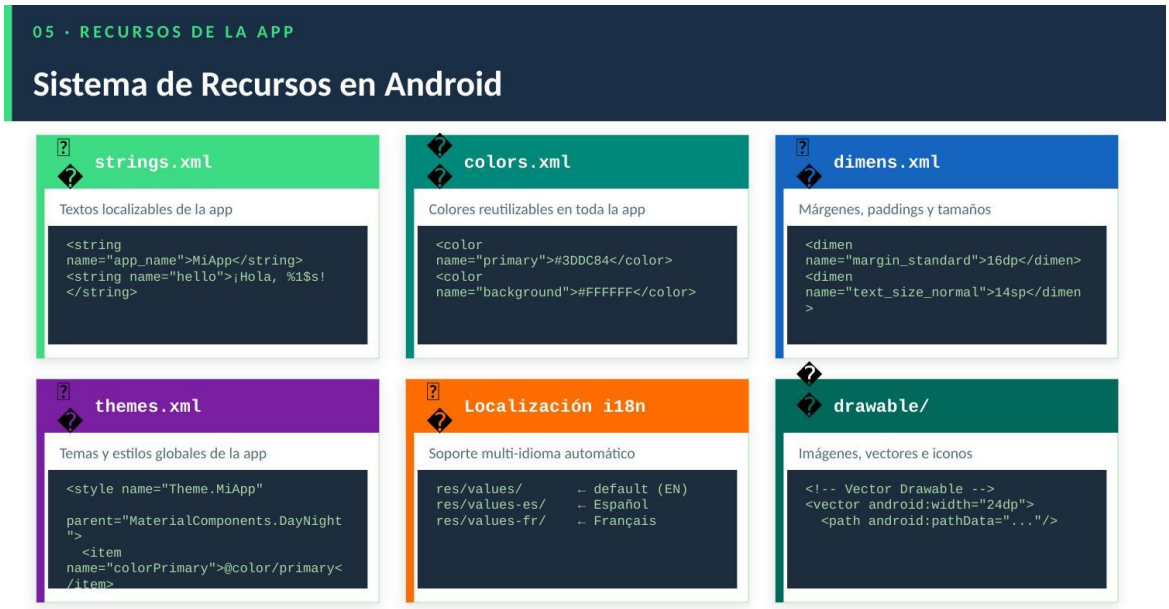
- **Componentes Material:** MaterialButton, TextInputLayout, CardView, BottomNavigationView, Snackbar, Chip...
- **Colores dinámicos:** Android 12+ genera paletas de color desde el fondo de pantalla del usuario
- **Documentación:** material.io/design — referencia completa de componentes y directrices

Diseño Responsivo

Android necesita adaptarse a múltiples tamaños de pantalla. Se usa el sistema de calificadores de recursos:

```
res/layout/                    # Layout por defecto (teléfono)
res/layout-sw600dp/            # Tablets de 7" (smallest width 600dp)
res/layout-sw720dp/            # Tablets de 10"
res/layout-land/               # Orientación horizontal
res/values-sw600dp/dimens.xml # Dimensiones específicas para tablet
```

5. Recursos de la Aplicación



5.1. Estructura del Directorio de Recursos

Todos los recursos de la app se ubican en `app/src/main/res/` y se acceden desde código mediante la clase `R`:

```
res/
├── values/       ← strings.xml, colors.xml, themes.xml, dimens.xml
├── drawable/     ← Imágenes, vectores, shapes XML
├── mipmap/       ← Iconos de la app (diferentes densidades)
├── layout/       ← Ficheros XML de layout
├── menu/         ← Menús XML
├── anim/         ← Animaciones
├── raw/          ← Ficheros sin procesar (audio, JSON...)
└── xml/          ← Ficheros XML misceláneos
```

Acceder a recursos desde código Kotlin

```
val texto  = getString(R.string.app_name)
val color  = getColor(R.color.primary)
val margen = resources.getDimensionPixelSize(R.dimen.margin_standard)
val icono  = getDrawable(R.drawable.ic_logo)
val vista  = layoutInflater.inflate(R.layout.activity_main, null)
```

5.2. Strings (strings.xml)

Todos los textos visibles de la app deben definirse en strings.xml para facilitar la localización:

```
<resources>
    <string name="app_name">Mi Aplicación</string>
    <!-- String con parámetro -->
    <string name="hello">¡Hola, %1$s!</string>
    <!-- Array de strings -->
    <string-array name="paises">
        <item>España</item>
        <item>México</item>
    </string-array>
    <!-- Plurales -->
    <plurals name="num_mensajes">
        <item quantity="one">%d mensaje</item>
        <item quantity="other">%d mensajes</item>
    </plurals>
</resources>
```

Uso en código

```
val nombre   = getString(R.string.app_name)
val saludo   = getString(R.string.hello, "Juan")  // → "¡Hola, Juan!"
val paises   = resources.getStringArray(R.array.paises)
val mensajes = resources.getQuantityString(R.plurals.num_mensajes, 5, 5)
```

5.3. Colores (colors.xml)

```
<resources>
    <color name="primary">#6200EE</color>
    <color name="primary_dark">#3700B3</color>
    <color name="secondary">#03DAC6</color>
    <!-- Con transparencia (formato ARGB) -->
    <color name="semi_transparent">#80000000</color>
</resources>
```

5.4. Dimensiones (dimens.xml)

Las dimensiones se definen en `dp` (density-independent pixels) para layouts y en `sp` (scale-independent pixels) para textos. Esto garantiza la consistencia visual en todas las densidades de pantalla:

```
<resources>
    <dimen name="margin_small">8dp</dimen>
    <dimen name="margin_standard">16dp</dimen>
    <dimen name="margin_large">24dp</dimen>
    <dimen name="text_size_normal">14sp</dimen>
    <dimen name="text_size_title">18sp</dimen>
    <dimen name="button_height">48dp</dimen>
    <dimen name="card_corner_radius">8dp</dimen>
</resources>
```

5.5. Temas y Estilos (themes.xml)

Los temas definen el aspecto visual global de la aplicación. Se basan en Material Components:

```
<resources>
    <style name="Theme.MiApp"
        parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <item name="colorPrimary">@color/primary</item>
        <item name="colorPrimaryVariant">@color/primary_dark</item>
        <item name="colorOnPrimary">@color/white</item>
        <item name="colorSecondary">@color/secondary</item>
    </style>
</resources>
```

5.6. Internacionalización (i18n)

Android soporta de forma nativa múltiples idiomas mediante carpetas de recursos con calificador de idioma:

```
res/values/         ← Idioma por defecto (inglés)
res/values-es/      ← Español
res/values-fr/      ← Francés
res/values-de/      ← Alemán
res/values-zh/      ← Chino
res/values-pt-rBR/  ← Portugués de Brasil
```

Android selecciona automáticamente los recursos según el idioma configurado en el dispositivo. Solo es necesario duplicar los ficheros strings.xml en cada carpeta.

5.7. Recursos Gráficos (drawable)

Vector Drawable (drawable/ic_check.xml)

Los vectores son la forma recomendada de definir iconos: escalan perfectamente a cualquier densidad de pantalla:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24">
    <path
        android:fillColor="@color/primary"
        android:pathData="M9,16.17L4.83,12l-1.42,1.41L9,19 21,7l-1.41,-
1.41z"/>
</vector>
```

Shape Drawable (fondo_redondeado.xml)

Permite crear fondos y formas sin necesidad de imágenes bitmap:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:shape="rectangle">
    <solid android:color="@color/primary"/>
    <corners android:radius="@dimen/card_corner_radius"/>
    <padding android:left="16dp" android:top="8dp"
             android:right="16dp" android:bottom="8dp"/>
</shape>
```

6. Resumen y Actividades

6.1. Puntos Clave del Tema

> *Android nació en 2003 (Andy Rubin) y fue adquirido por Google en 2005. Hoy cuenta con más de 3.000 millones de usuarios activos, representando ~70% del mercado global de smartphones.*

> *La arquitectura de Android se organiza en 5 capas: Aplicaciones, Framework de Aplicaciones, Librerías Nativas + ART, HAL y Kernel Linux. ART usa compilación AOT+JIT para maximizar el rendimiento.*

> *Android Studio es el IDE oficial. El SDK Manager gestiona versiones y componentes. ADB (Android Debug Bridge) es la herramienta esencial para depurar tanto en emulador como en dispositivo real.*

> *Una app Android se construye con 4 tipos de componentes: Activity (UI), Service (background), BroadcastReceiver (eventos del sistema) y ContentProvider (datos compartidos). Todos se declaran en el AndroidManifest.xml.*

> *Los Intents conectan componentes (explícitos = misma app, implícitos = sistema elige). Los Layouts organizan la UI (ConstraintLayout recomendado). Material Design 3 es el lenguaje visual de Android.*

6.2. Actividades Prácticas

15. **Instalación:** Instalar Android Studio y crear el primer proyecto 'Hello World'. Ejecutarlo en el emulador.
16. **Exploración:** Navegar por la estructura del proyecto. Abrir y estudiar AndroidManifest.xml y build.gradle.kts.
17. **Emulador:** Crear un AVD con configuración Pixel 7 / API 34. Lanzar la app y explorar las opciones de depuración.
18. **UI Básica:** Diseñar un layout con ConstraintLayout que incluya un TextView, un EditText, un Button y un ImageView.
19. **Recursos:** Crear strings.xml con al menos 5 strings y colors.xml con la paleta de colores de la app. Referenciarlos desde el layout.
20. **Ciclo de Vida:** Implementar Log.d() en todos los métodos del ciclo de vida (onCreate, onStart, onResume, onPause, onStop, onDestroy) y observar el orden en Logcat.
21. **Intents:** Crear una app con tres botones que abran respectivamente: el navegador web, el marcador telefónico y el selector de compartir contenido.

6.3. Recursos Adicionales

Documentación oficial:

- Android Developers — Documentación oficial de Android
- Android Studio — Descarga oficial
- Material Design Guidelines — Sistema de diseño de Google
- Android API Levels — Referencia de versiones y niveles de API
- Kotlin Documentation — Documentación oficial del lenguaje Kotlin