



ITESO, Universidad  
Jesuita de Guadalajara


# Taller de tecnología móvil

- Android
- Kotlin

# ***Temario***

## Módulo 1

- Introducción al sistema operativo Android
- Lenguajes de programación de Android
- Programación básica
- Entorno de desarrollo Android (IDE)
- Estructura de una aplicación en Android



# Introducción al sistema operativo Android

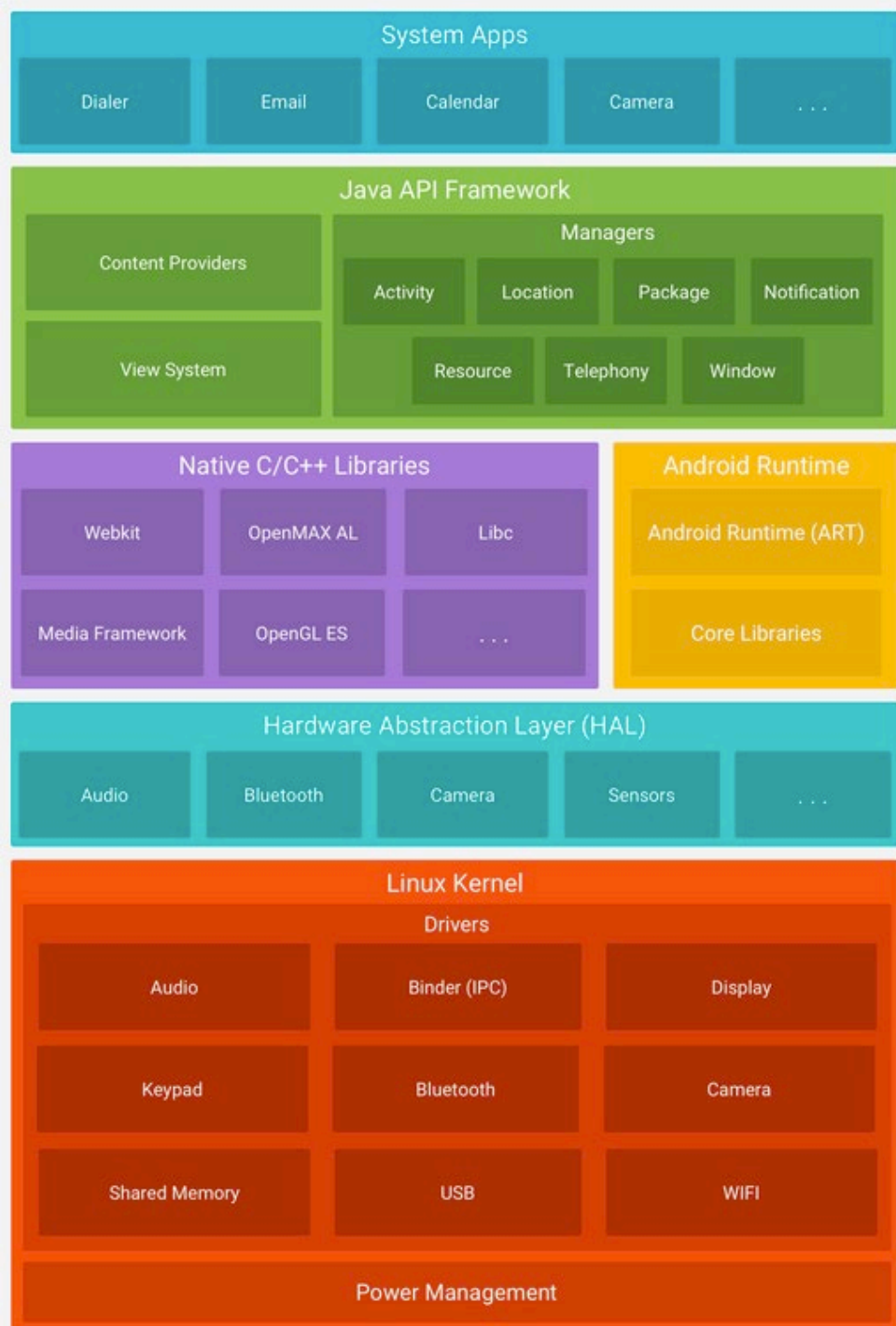
# ***Introducción al sistema operativo Android***

## **¿Qué es Android?**

**Android** es un sistema operativo basado en **Linux** que fue pensado en un principio para usarse con un teclado y un cursor que permitía navegar por las aplicaciones. En la actualidad, tras varias actualizaciones, está pensado para dispositivos móviles con pantalla táctil, ya sean smartphones o tabletas. Inicialmente fue creado por la compañía de software Android Inc, pero en el año 2005 Google compró la empresa y 2 años después presentó el sistema operativo.



# Arquitectura Android



- Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama, se muestran los componentes principales de la plataforma Android.

# Requerimientos de software y hardware

## Requerimientos de software

### Sistema Operativo

- Windows 64-bit Microsoft® Windows® 8/10
- Mac OS X 10.4.8 o posterior (solo X86)
- Linux (Testeado en Linux Ubuntu Dapper Drake)

### IDE de desarrollo

- Android Studio
- Visual Studio Code

## Requerimientos de hardware

2GB Mínimo de memoria RAM

Más de 1 GB disponible en Disco Duro que será utilizado para Android SDK, imágenes de emuladores y caches

Resolución mínima de pantalla de 1280 x 800 px

Dispositivo Android o simulador con Sistema operativo versión 4.4 KitKat como mínimo



# ***Lenguajes de programación de Android***

# ¿Qué es Kotlin?

**Kotlin** es un proyecto gratuito y de código abierto registrado bajo la licencia de Apache 2.0. El código del proyecto se desarrolla abiertamente en GitHub y está a cargo, principalmente, del equipo empleado en JetBrains, con contribuciones de Google y otros. Con **Kotlin** se logra reducir de manera significativa la cantidad de líneas de código dentro de una aplicación.

```
KOTLIN

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        fab.setOnClickListener { view ->
            Snackbar.make(view, "Hello $name", Snackbar.LENGTH_LONG).show()
        }
    }
}
```

Nullable and NonNull types help reduce NullPointerExceptions

Use lambdas for concise event handling code

Use template expressions in strings to avoid concatenation

Semicolons are optional



# XML

**XML** por su facilidad de uso, y gran potencial en el proceso, es el complemento perfecto para el desarrollo de aplicaciones Android; permite crear las interfaces de usuario, generar y configurar archivos de gran utilidad para el correcto funcionamiento de la aplicación, permite en modo diseño, crear los elementos necesario como; botones, cajas de texto, etiquetas de texto, contenedores de imágenes, etc.; todo esto mediante etiquetas de apertura <> y cierre </>.

```
<?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>
```



# ***Programación básica***

# Kotlin Playground

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    val evento : String = "Ada Byron"
    println("Hola $evento")
}
```

Hola Ada Byron

# Variables

Una **variable** no es más que un espacio de memoria en el que podemos guardar información. Una variable se compone de:

- Nombre o identificador
- Tipo de dato
- Valor
- Ámbito

```
1.  var numeroFavorito = 1
```

```
1.  var numeroFavorito: Int = 1
```

Kotlin utiliza dos palabras clave diferentes para declarar variables: **val** y **var**.

- Usa **val** para una variable cuyo valor no cambia nunca. No puedes volver a asignar un valor a una variable que se declaró mediante **val**.
- Usa **var** para una variable cuyo valor puede cambiar.

```
val languageName: String = "Kotlin"
```

# Trabajando con variables

```
1. fun main(args: Array<String>) {  
2.     var a = 10  
3.     var b = 5  
4.  
5.     print("Suma: ")  
6.     println(a + b)  
7.  
8.     print("Resta: ")  
9.     println(a - b)  
10.  
11.    print("Multiplicación: ")  
12.    println(a * b)  
13.  
14.    print("División: ")  
15.    println(a / b)
```



```
print("El módulo (resto): ")  
println(a % b)
```

Ver práctica No. 1

# Comentarios

Para realizar un comentario de una sola línea, usa barra doble **//** y seguidamente escribes las palabras que proporcionen el contexto en el lugar del código.

```
fun main() {  
    // Ejemplo de suma de dos números  
    val sumOfTwoNumbers: Int // (1)  
  
    val firstNumber = 1 // (2)  
    val secondNumber = 5 // (3)  
  
    sumOfTwoNumbers = firstNumber.plus(secondNumber) // (4)  
    println("($firstNumber + $secondNumber) = $sumOfTwoNumbers") // (5)  
}
```

Si requieres insertar un comentario que contenga múltiples línea, entonces abre con barra-asterisco, escribes *n* cantidad de líneas y cierra con asterisco-barra. **/\* ... \*/**

```
/*  
 * Copyright 2021 James Revelo. Todos los derechos reservados.  
 * Ver Copyright.txt para más detalles de permisos  
*/
```

# Funciones

Las **funciones** se declaran usando la palabra clave **fun**, seguida del nombre del método, los paréntesis donde declararemos los valores de entrada y unas llaves que limitan la función.

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

!fun main(args: Array<String>) {
    showMyName()
    showMyLastName()
    showMyAge()
}

fun showMyName(){
    println("Me llamo Aris")
}

fun showMyLastName(){
    println("Mi Apellido es Guimerá")
}

fun showMyAge(){
    println("Tengo 24 años")
}
```

```
Me llamo Aris
Mi Apellido es Guimerá
Tengo 24 años
```

# ***Funciones***

Ver la práctica 2 del cuaderno de prácticas



# Controles de flujo

Las instrucciones condicionales nos permiten realizar lógica en función del resultado de una variable o condición.

## If- else

- La condición **if** es de las más habituales y realizará una función o varias solo si la condición que hemos generado es verdadera.

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    var name = "Aris"

    if(name == ("Aris")){
        println("Se llama Aris")
    }else{
        println("No se llama Aris")
    }
}
```

Se llama Aris

# ***Controles de flujo***

Ver la práctica 3 del cuaderno de prácticas

# Controles de flujo

## When

- **When** nos permite realizar una o varias acciones dependiendo del resultado recibido. También se podría hacer con el *if-else*.

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    var x = 3
    when (x) {
        1 -> print("x == 1")
        2 -> print("x == 2")
        else -> {
            print("x no es ni 1 ni 2")
        }
    }
}
```

x no es ni 1 ni 2

# Controles de flujo

Ver la práctica 4 del cuaderno de prácticas

# Controles de flujo

## For

- Para el control de flujo también existen los bucles. Un bucle **for** repite una operación en función del iterador que proporciones.
- **For** es el más utilizado.

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    for (i in 1..3) {
        println(i)
    }
    println("DownTo")
    for (i in 6 downTo 0 step 2) {
        println(i)
    }
}
```

```
1
2
3
DownTo
6
4
2
0
```

# Controles de flujo

Ver la práctica 5 del cuaderno de prácticas

# Arrays

Las **arrays** son secuencias de datos, del mismo tipo e identificados por un nombre común.

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    val weekDays = arrayOf("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo")

    for (weekDay in weekDays) {
        println(weekDay)
    }
}
```

```
Lunes
Martes
Miércoles
Jueves
Viernes
Sábado
Domingo
```

# Arrays

Ver la práctica 6 del cuaderno de prácticas



# Listas

Vimos todo el potencial que tenían los arrays, pero uno de los mayores inconvenientes era la limitación al definirlos, puesto que teníamos que saber de ante mano el tamaño de dicho array. Las **listas** se pueden clasificar en dos grandes grupos, las mutables e inmutables. Es decir, las que se pueden editar (mutables) y las que son solo de lectura (inmutable).

## *Inmutable - List<Type>*

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */

fun main() {
    val readOnly: List<String> = listOf("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo")

    readOnly.size //Muestra el tamaño de la lista
    readOnly.get(3) //Devuelve el valor de la posición 3
    readOnly.first() //Devuelve el primer valor
    readOnly.last() //Devuelve el último valor
    println(readOnly) //[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo]
}
```

[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo]

# Listas

Ver la práctica 7 del cuaderno de prácticas

# Listas

## *Mutable – MutableList<Type>*

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    var mutableList: MutableList<String> = mutableListOf("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado")

    println(mutableList) //[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado]

    mutableList.add("Domingo")

    println(mutableList) //[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo]
}
```

```
[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado]
[Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo]
```

# Listas

Ver la práctica 8 del cuaderno de prácticas



# Entorno de desarrollo Android (IDE)

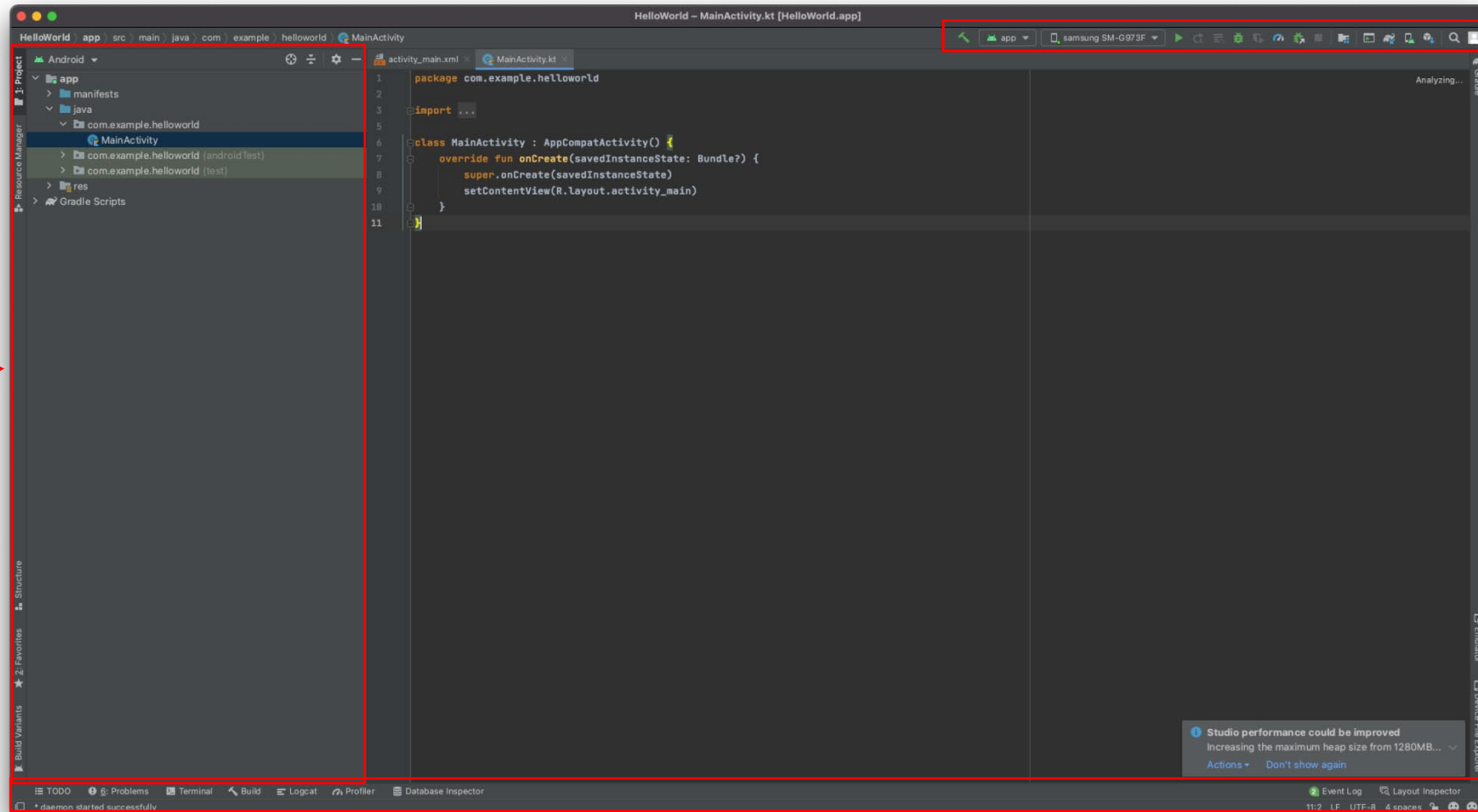
# Android Studio (IDE)

**Android Studio** es un entorno de desarrollo integrado (IDE) oficial para crear aplicaciones móviles en plataforma Android. Es un IDE gratuito para sistemas operativos como Microsoft Windows, Mac OS X y GNU/Linux.



# Android Studio (IDE)

Estructura  
de la  
aplicación

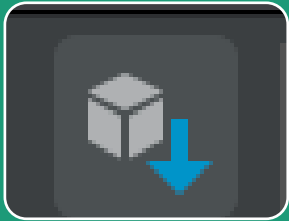


Aquí  
encontrarás el  
botón Run,  
debug,  
selección de  
dispositivos  
donde correr  
nuestra  
aplicación,  
SDK, AVD,  
entre otros.

Aquí encontrarás el DDMS donde veremos el proceso de  
nuestra aplicación

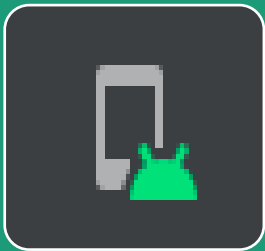
# Android SDK y AVD Manager

## Android SDK manager



El SDK (Software Development Kit) Manager, es una herramienta de Google gratuita indispensable, que permite administrar y descargar, librerías, utilerías, drivers, imágenes (emuladores), recursos, servicios, documentación, etc., para desarrollar aplicaciones en todas las versiones disponibles (API's) en plataforma Android.

## Android AVD Manage



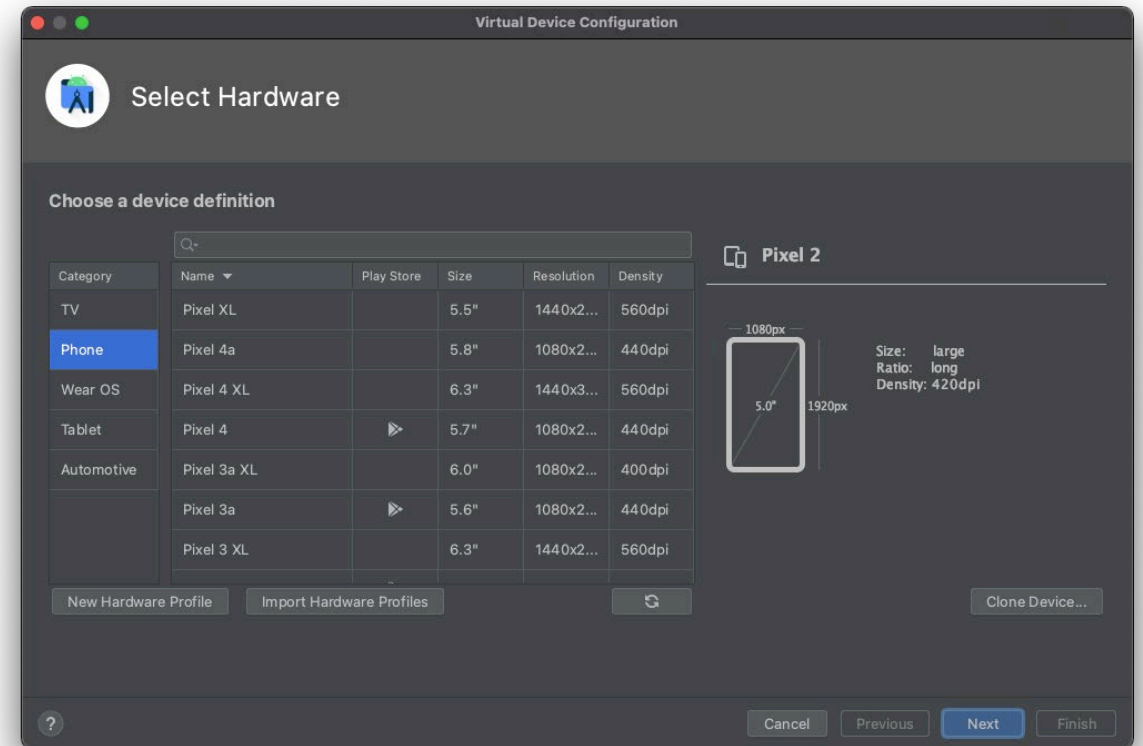
- Android AVD (Android Virtual Device) Manager, es una herramienta que permite definir, crear y administrar dispositivos virtuales para emular, ver y probar el funcionamiento del desarrollo de una aplicación.



# Android SDK y AVD Manager

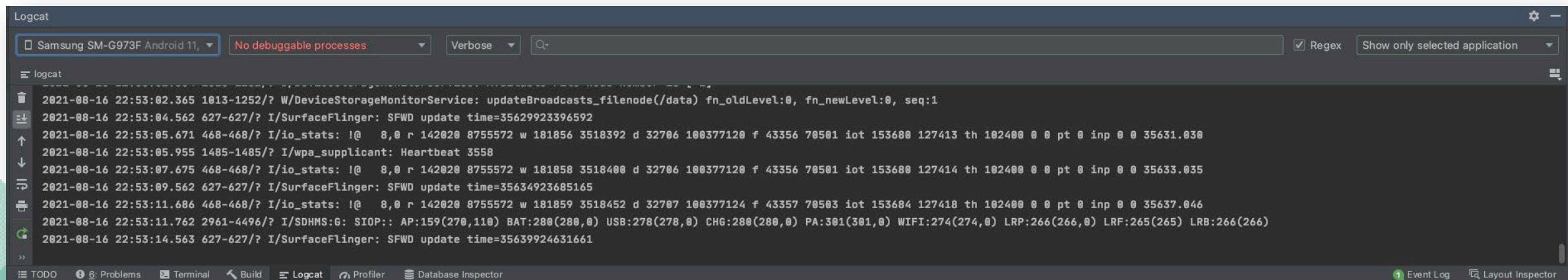
## Administración de AVD's

La **administración AVD**, permite elegir un emulador ya pre-configurado que se instalan desde el SDK o elegir un emulador personalizado ya previamente configurado, proporcionar información general sobre el emulador como lo es el Nombre del AVD, API, Microprocesador, entre otras.



# El panel DDMS

DDMS es un monitor de servicios para la depuración de procesos en aplicaciones Android, proporciona servicios de redireccionamiento de puertos, captura de pantallas, listado de dispositivos (tanto AVD's como físicos), hacer logcats, ver procesos, entre otras. Esta herramienta de monitoreo es muy útil para el desarrollo.

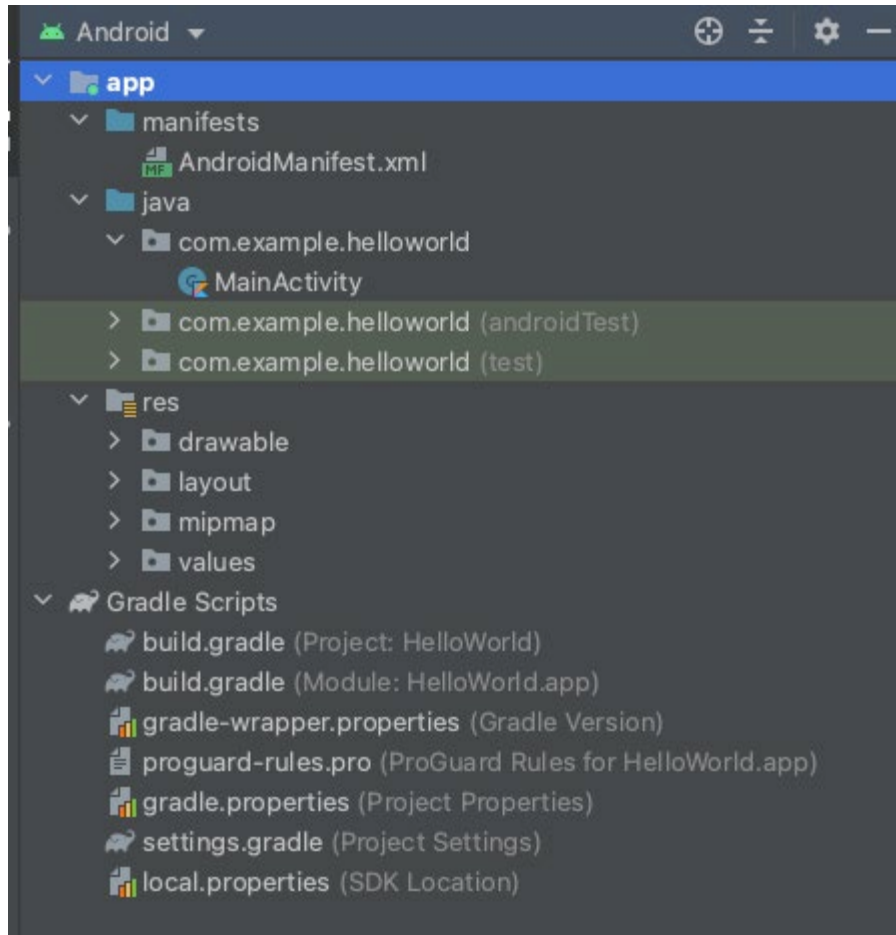




# Estructura de una aplicación en Android

# Estructura

Antes de iniciar a programar para Android, es importante conocer la estructura de una aplicación. Cuando se crea un nuevo proyecto en Android Studio, éste, genera una estructura jerárquica de archivos, módulos y carpetas que ayuda a organizar, definir y configurar los parámetros, la lógica y las pantallas (a nivel de usuario) para el funcionamiento de las aplicaciones durante y después del desarrollo.





/manifest

Esta carpeta contiene un archivo que contiene nodos descriptivos sobre las características de la aplicación, tales como: permisos de usuario, pantallas que soportará la aplicación, el nombre del paquete de Java, las actividades (activities), servicios, icono de la aplicación, entre otros.



/res

Contiene todos los recursos gráficos y de texto para el proyecto.



/drawable

Se encuentran las imágenes en formato JPEG o PNG, Google tiene preferencia por esta última; para mayor organización se recomienda definir subcarpetas para las diferentes densidades de pantalla.



/layout

Contiene los archivos de definición .xml de las diferentes pantallas de interfaz gráfica que se mostrarán en la aplicación, tanto orientación vertical como horizontal.



/mipmap

Contiene las imágenes y otros elementos gráficos usados por la aplicación, tales como, el icono de la aplicación. Se puede definir diferentes recursos gráficos dependiendo de la resolución y densidad de pantalla en subcarpetas.



/values

Contiene los valores del proyecto, definiendo los nombres de las variables que serán referenciadas en el código XML o java respectivamente. Estas variables pueden ser cadenas de texto, definición de colores, dimensiones, tipos de datos (int, bool, string, etc), incluso valores constantes.



Strings.xml

Archivo XML que define cadenas de texto usadas en la aplicación. Por ejemplo, para colocar los títulos de las ventanas o el nombre de la aplicación y referenciarlas cuando se necesite en el código xml o java.



colors.xml

Contiene todos los recursos gráficos y de texto para el proyecto.



themes.xml

Se encuentran las imágenes en formato JPEG o PNG, Google tiene preferencia por esta última; para mayor organización se recomienda definir subcarpetas para las diferentes densidades de pantalla.



app/build.gradle

Contiene los archivos de definición .xml de las diferentes pantallas de interfaz gráfica que se mostrarán en la aplicación, tanto orientación vertical como horizontal.



Hola Mundo

# Ejercicio

Ver la práctica 9 del cuaderno de prácticas

```
fun main(args: Array<String>) {  
    println("Hello, World!")  
}
```



# Dudas

***"Nada en la vida debe ser temido, solamente comprendido. Ahora es el momento de comprender más para temer menos."* - Marie Curie**

Primera mujer en recibir el Premio Nobel y la primera persona en ganar el Premio Nobel en dos categorías diferentes. El primer premio fue por la investigación de la radiactividad y el segundo por su trabajo en la química.

# Referencias

- Android Studio. Descargar. Recuperado de <https://developer.android.com/studio>
- Kotlinlang. Documentación. Recuperado de <https://kotlinlang.org/docs/home.html>
- Kotlinlang. Funciones. Recuperado de <https://kotlinlang.org/docs/functions.html>
- El sistema operativo Android. Recuperado de <https://www.rastreator.com/telefonía/articulos-destacados/el-sistema-operativo-android.aspx>
- Documentación para desarrolladores de apps. Recuperado de <https://developer.android.com/docs?hl=es-419>