U1. INTRODUCCIÓN A PMDM

- Android developers:
  - https://developer.android.com/?hl=es-419
- ¿Cómo desarrollo una app móvil?
  - **Android Studio**
- ¿Y en qué lenguaje de programación?
  - Kotlin

Razones para desarrollar con Kotlin

- Recomendado por Google desde el 2017. Oficial.
- Simpleza. ¿Cómo se lleva Java con los valores nulos?
- Multiplataforma
- 100% Interoperable con Java. Kotlin usa la JVM (máquina virtual de Java)
- Integración total en Android Studio
- JetBrains lo recomienda (empresa que desarrolla IntelliJ)

U1. Instalación de Android Studio

- No emplear máquinas virtuales
- Instalar en una máquina física

- ¿Por qué digo esto?

U1. Instalación de Android Studio

1. Rendimiento
2. Emulador Android
3. Limitaciones HW
4. Problemas con la virtualización

U1. Hola Mundo con Android Studio

1. Analiza y trastea este nuevo IDE
2. Crea y personaliza un botón con tu nombre y un color de fondo
3. Al pulsar el botón debe escribirse un: “Hola Mundo”
4. https://developer.android.com/get-started/overview?hl=es-419
5. ¿Qué es gradle?

U1. Android Studio. Teoría

Minimum SDK: es la versión de mínima de Android que deben tener los dispositivos para poder ejecutar la app. Para soportar el máximo de dispositivos que sea posible, se debe establecer a la versión más baja posible. Sin embargo, cuanto más baja es la versión menos características de android modernas puede utilizar la app.

U1. Aspectos fundamentales

https://developer.android.com/guide/components/fundamentals?hl=es-419

A continuación, los más importantes

U1. Aspectos fundamentales

- **AndroidManifest.xml** El archivo de manifiesto describe las características fundamentales de la aplicación y define cada uno de sus componentes. Aquí especificaremos las Activities que tiene nuestra app y los permisos que requiere (Cámara, Contactos, Internet, etc.)

U1. Aspectos fundamentales

- **MainActivity.kt** Contiene la definición de clase para la Activity (pantalla) principal. En este fichero programaremos el comportamiento de esta Activity.

- **activity_main.xml** Este archivo XML define el layout de la actividad, es decir los elementos (widgets) que tiene la pantalla. Por defecto, contiene un elemento TextView con el texto ¡Hola mundo!.

U1. Aspectos fundamentales

- **gradle** Android Studio utiliza Gradle para compilar y construir la aplicación. Hay un archivo **build.gradle** para cada módulo del proyecto, así como un archivo **build.gradle** de todo el proyecto. Por lo general, sólo estaremos interesados en el archivo **build.gradle (module: app)**. Ahí es donde se encuentran las dependencias de construcción de la aplicación, incluyendo los ajustes **defaultConfig**

U1. Interfaces de usuario

El SDK de Android contiene componentes y *widgets* que facilitan el diseño de la interfaz de usuario, la comunicación y la interacción con las ***Activities*** creadas para la app.

U1. Interfaces de usuario

- **View**. Dentro del SDK hay un paquete llamado **android.view** que contiene tanto interfaces como clases utilizadas para dibujar en pantalla. Es la clase base para los widgets y layouts
- **Widget.** El paquete **android.widget** que se deriva de la clase View. Los widgets representan clases dentro de ese paquete y contienen casi cualquier cosa que queramos dibujar, como **imageView, FrameLayout, EditText** y los objetos **Button**

U1. Interfaces de usuario

- **Layout.** Es un *widget* particular que se encuentra en el paquete **android.widget.** Es un **objeto** View, no dibuja nada en pantalla, pero actúa como un contenedor (padre) que soporta otros *widgets* (hijos). Así, su función es definir el lugar de la pantalla en el que se dibujarán los *widgets*, de acuerdo a determinadas reglas

U1. Directorios

- **drawable/** Directorio de imágenes.
- **layout/** Directorio para los archivos que definen la interfaz de usuario de la aplicación, como activity_main.xml, que describe un diseño básico para la MainActivity.
- **mipmap/** Contiene los iconos Lanzadores.
- **values/** Directorio para otros archivos XML que contienen una colección de recursos.

U1. Código básico

- **MainActivity** hereda de **AppCompatActivity**: **AppCompatActivity** es una clase que proporciona compatibilidad con características más nuevas de Android en versiones más antiguas.

- **onCreate()**: Este método es el primero que se ejecuta cuando la actividad se crea. Aquí es donde configuramos la interfaz de usuario y definimos las acciones.

U1. Código básico

- **findViewById:** Nos permite encontrar elementos de la interfaz visual (XML) para interactuar con ellos desde el código.

- **setOnClickListener**: Son métodos que "escuchan" cuando el usuario presiona un botón, y ejecutan el código dentro de la lambda correspondiente.

U1. Código básico

- **R.layout.activity_main**:
  - **R**: Es la clase que agrupa todos los recursos.
  - **layout**: Es una subclase dentro de R que contiene referencias a todos los archivos de diseño XML que están en la carpeta res/layout/.
  - **activity_main**: Es el nombre del archivo XML que define el diseño de la interfaz de la actividad principal. Este nombre viene del archivo activity_main.xml que está en la carpeta res/layout/.

U1. Código básico

- La clase **R:** es el puente que te permite acceder fácilmente a todos los recursos de tu aplicación Android desde el código. Por ejemplo:

    - Si agregas una imagen llamada logo.png en res/drawable/, la clase R creará algo como R.drawable.logo.
    - Si defines un string en res/values/strings.xml, se generará R.string.tu_string.

U1. Código básico

- **ContextCompat** es una clase que te facilita usar recursos y funciones de **forma compatible con versiones anteriores** de Android.
- Ejemplo sencillo:
  - Obtiene el color compatible con versiones antiguas de Android
    - *val color = ContextCompat.getColor(context, R.color.mi_color)*

U1. Código básico

- **ContextCompat** ayuda a **evitar** tener que hacer **verificaciones manuales** de la versión del sistema operativo.
- Ejemplo avanzado:
  - Verifica si el permiso ha sido concedido
    - *if(ContextCompat.checkSelfPermission(context, Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED)*

      { // Puedes acceder a la cámara }

U1. Código básico. Clase ViewGroup

Existe una clase especial derivada de View llamada **ViewGroup**, la cual permite mostrar los objetos View en la pantalla de forma organizada.

La clase **ViewGroup** puede contener subclases, las cuales se puede programar por el método addView().

Si utilizamos recursos XML, estos se añaden definiendo el objeto hijo como un nodo XML.

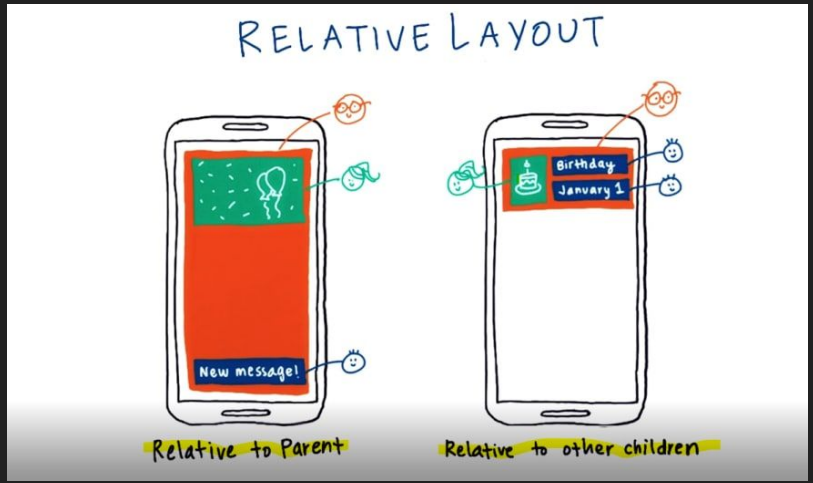
Las ***subclases directas*** derivadas de la clase ViewGroup son aquellas que terminan en Layout, y sirven para posicionar *widgets* en la pantalla.

U1. Subclases directas. ViewGroup

- **LinearLayout**. alinea a todos los hijos secuencialmente, ya sea de forma horizontal o vertical

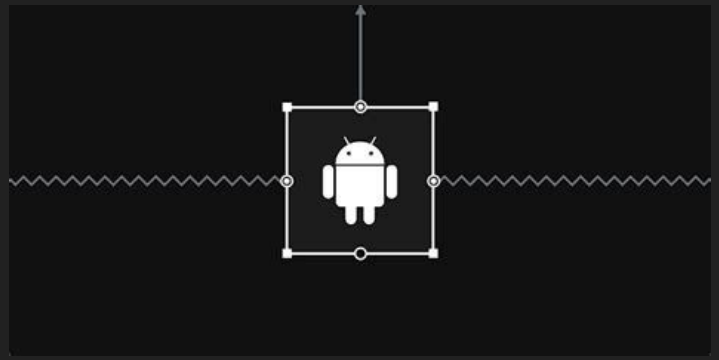
U1. Subclases directas. ViewGroup

- **RelativeLayout**. (Deprecado, reemplazado por ConstraintLayout en muchos casos). Permite posicionar elementos relativos a otros componentes o a los bordes del contenedor.



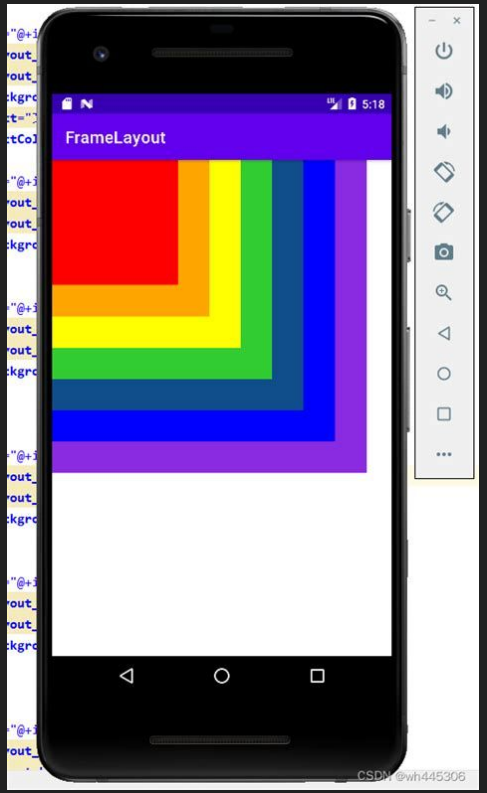
U1. Subclases directas. ViewGroup

- **ConstraintLayout**. (El layout más flexible y recomendado para crear interfaces complejas. Permite posicionar elementos de manera precisa usando constraints (restricciones) relativas a otros elementos o al contenedor.



U1. Subclases directas. ViewGroup

- **FrameLayout**. (Organiza los elementos en capas, apilándolos uno encima de otro. Útil para contener un solo elemento o elementos superpuestos.



U1. Subclases directas. ViewGroup

- **TableLayout**. (Organiza elementos en filas y columnas, similar a una tabla HTML. Cada fila es un TableRow.

U1. Práctica 1. App contador

1. La aplicación deberá tener:
   a. Un **TextView** que mostrará un número.
   b. Dos **Button**: uno para incrementar y otro para decrementar el valor del contador.
2. El contador debe comenzar en 0 y debe aumentar o disminuir con cada pulsación de los botones.
3. Si el valor del contador llega a 10, se debe cambiar el color del TextView a verde. Si baja a -10, el color debe ser rojo.

U1. Práctica 2. Eventos y Seekbar

1. La aplicación deberá tener:
    a. Dos botones: uno para incrementar y otro para decrementar un valor.
    b. Un TextView que muestre el valor actual.
    c. Un **SeekBar** que permita ajustar el valor dentro de un rango específico de manera manual.
    d. Utiliza **ConstraintLayout**

U2. Jetpack Compose

Jetpack Compose es un kit de herramientas moderno para crear IU nativas de Android. Jetpack Compose simplifica y acelera el desarrollo de IU en Android con menos código

Jetpack Compose usa un complemento de compilador de Kotlin para transformar estas funciones que admiten composición en elementos de la IU de la app.

U2. Jetpack Compose

La anotación **@Preview** te permite obtener una vista previa de tus funciones que admiten composición dentro de Android Studio sin tener que compilar e instalar la app en un emulador o dispositivo Android.

U2. Jetpack Compose. Diseños

Los elementos de la IU son jerárquicos, ya que unos contienen a otros. En Compose, compilas una jerarquía de la IU llamando a las funciones que admiten composición desde otras funciones del mismo tipo.

U2. Jetpack Compose. Column, Row, Box

La función **Column** te permite organizar los elementos de forma vertical. Agrega el objeto **Column** a la función **MessageCard**.

Puedes usar **Row** para organizar los elementos de manera horizontal y **Box** para apilarlos.

U2. Jetpack Compose. Modificadores

Para decorar o configurar un elemento componible, Compose usa **modificadores**, que te permiten cambiar su **tamaño**, **diseño** y **apariencia**, o agregar **interacciones** de alto nivel, como hacer que un elemento sea apto para recibir clics.

U2. Jetpack Compose. Material Design

Material Design se basa en tres pilares: **Color**, **Typography** y **Shape.**

Proporciona una estructura sólida y componentes de UI ya optimizados para crear aplicaciones modernas, consistentes y accesibles, mientras permite flexibilidad para personalizar el diseño de acuerdo con las necesidades de tu aplicación.

U2. Jetpack Compose. Material Design

Material Design ofrece una biblioteca por defecto llamada **androidx.compose.material** que incluyen componentes preconstruidos

También se pueden personalizar mediante el sistema de **MaterialTheme** permite que configures los colores, las formas y las tipografías de tu aplicación para ajustarse a tu propio estilo.

U2. Jetpack Compose. Material Design

Colección de componentes de UI listos para usar en Jetpack Compose, organizados en categorías como botones, cuadros de diálogo, menús, selectores de fecha y más. Estos componentes están basados en los principios de Material Design y otros elementos personalizados.

https://composables.com/ -> **Tarea1. Componentes**

U1. Práctica 3. Lista de tareas

Funcionalidades a implementar:

1. **Agregar tareas**. Los usuarios podrán añadir nuevas tareas a una lista.
2. **Marcar tareas como completadas**. Se podrá hacer clic en una tarea para marcarla como completada o eliminarla.
3. **Persistencia con SharedPreferences**. Al cerrar y abrir la app, la lista de tareas debe mantenerse.

U1. Práctica 3. Lista de tareas

Consejos / Requisitos:

1. **EditText**. Para ingresar nuevas tareas
2. **RecyclerView**. Para mostrar la lista de tareas
3. **Button**. Para añadir una tarea a la lista
4. **Persistencia.** Guardar la lista en SharedPreferences para que las tareas se mantengan después de cerrar la app.