

## Capítulo 2

### UD01. Android Studio

#### Resultados de avaliación

RA1. Aplica tecnoloxías de desenvolvemento para dispositivos móbiles, e avalía as súas características e as súas capacidades.

RA2. Desenvolve aplicacións para dispositivos móbiles, para o que analiza e emprega as tecnoloxías e as librerías específicas.

#### Criterios de avaliación

CA1.1 Analizáronse as limitacións que presenta a execución de aplicacións nos dispositivos móbiles.

CA1.2 Recoñecéronse os sistemas operativos empregados en dispositivos móbiles e as súas características.

CA1.3 Identificáronse as tecnoloxías de desenvolvemento de aplicacións para dispositivos móbiles.

CA1.4 Instaláronse, configuráronse e utilizáronse contornos de traballo para o desenvolvemento de aplicacións para dispositivos móbiles.

CA1.5 Identificáronse as configuracións en que se clasifican os dispositivos móbiles con base nas súas características.

CA1.6 Describíronse os perfís que establecen a relación entre o dispositivo e a aplicación.

CA1.7 Analizouse a estrutura de aplicacións existentes para dispositivos móbiles, e identificáronse as clases utilizadas.

CA1.8 Realizáronse modificacións sobre aplicacións existentes.

CA1.9 Utilizáronse emuladores para comprobar o funcionamento das aplicacións.

CA2.1 Xerouse a estrutura de clases necesaria para a aplicación.

CA2.8 Realizáronse probas de interacción entre o usuario e a aplicación para mellorar as aplicacións desenvolvidas a partir de emuladores.

CA2.9 Empaquetáronse e despregáronse as aplicacións desenvolvidas en dispositivos móbiles reais.

CA2.10 Documentáronse os procesos necesarios para o desenvolvemento das aplicacións.

BC1. Tecnoloxías de desenvolvemento para dispositivos móbiles, e avalía as súas características e as súas capacidades.

Limitacións que presenta a execución de aplicacións nos dispositivos móbiles: desconexión, seguridade, memoria, consumo de batería, almacenamento, etc.

Estrutura dunha aplicación para dispositivo móbil.

Modificación de aplicacións existentes.

Uso do contorno de execución do administrador de aplicacións.

Sistemas operativos empregados en dispositivos móbiles.

Tecnoloxías de desenvolvemento de aplicacións para dispositivos móbiles.

Contornos integrados de traballo.

Módulos para o desenvolvemento de aplicacións móbiles.

Emuladores.

Configuracións: tipos e características. Dispositivos soportados.

Perfís: características, arquitectura e requisitos. Dispositivos soportados.

Ciclo de vida dunha aplicación: descubrimento, instalación, execución, actualización e borrado.

Ferramentas e fases de construción.

Probas de interacción.

Empaquetaxe e distribución.

Documentación do desenvolvemento das aplicacións.

Estrutura de clases dunha aplicación.

Última actualización: 17.01.2024

Subseccións de UD01. Android Studio

Android

Android es un sistema operativo móvil desarrollado por Google. Fue diseñado inicialmente para dispositivos móviles como teléfonos inteligentes y tabletas, pero con el tiempo se ha expandido para incluir otros dispositivos como televisores inteligentes (Android TV), relojes inteligentes (Wear OS), sistemas de infoentretenimiento en automóviles (Android Auto) y más.



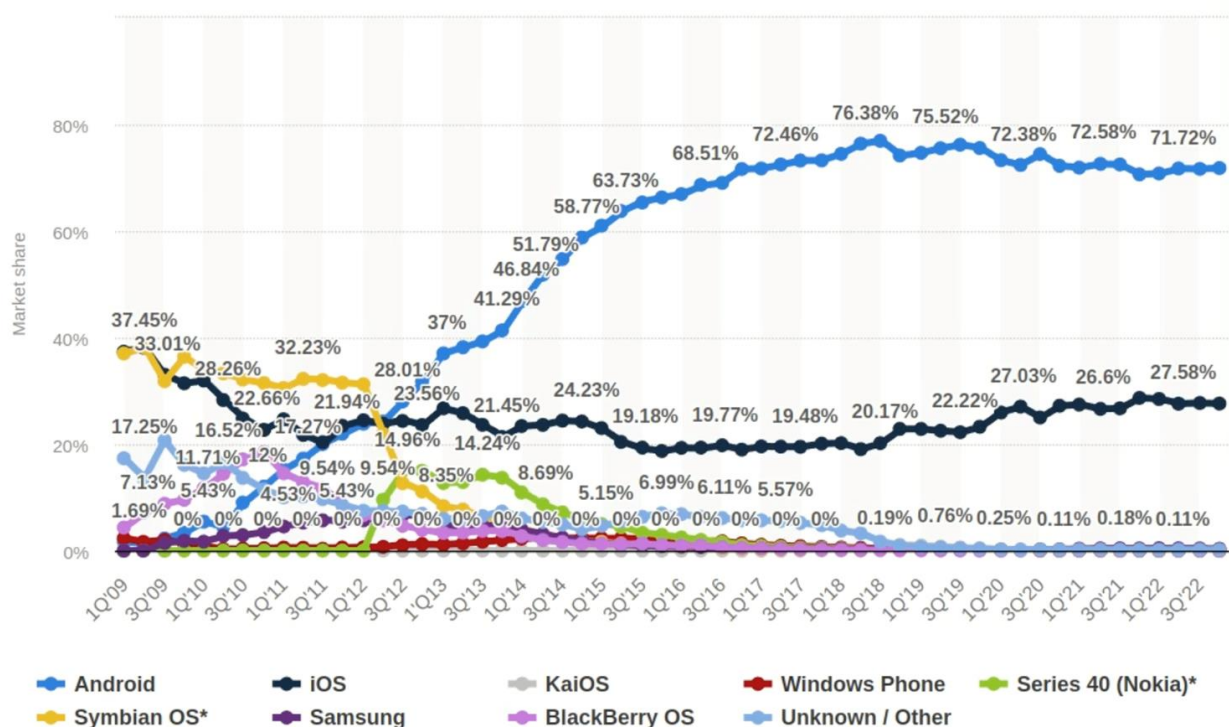
Android se basa en el núcleo de Linux y utiliza una arquitectura de código abierto, lo que significa que su código fuente está disponible públicamente y puede ser modificado y distribuido por otros desarrolladores bajo ciertas restricciones. Esto ha permitido que una amplia comunidad de desarrolladores y fabricantes contribuya al sistema operativo y cree sus propias versiones personalizadas.

Una de las características más destacadas de Android es su tienda de aplicaciones oficial, llamada Google Play Store, donde los usuarios pueden descargar e instalar una amplia variedad de aplicaciones, juegos, libros electrónicos y otros contenidos. Los desarrolladores pueden crear aplicaciones para Android utilizando el lenguaje de programación Java o, más recientemente, Kotlin.

#### Mercado

En el mundo de la programación móvil, Android y iOS son las dos principales plataformas que dominan el mercado de desarrollo de aplicaciones móviles. Ambas tienen un impacto significativo y atraen a una gran cantidad de desarrolladores. La elección entre desarrollar para Android o iOS a menudo depende de varios factores, como la audiencia objetivo, las preferencias de desarrollo y los objetivos comerciales.

En el gráfico siguiente, se ilustra la trayectoria evolutiva de los sistemas operativos móviles a nivel global, abarcando desde el primer cuatrimestre de 2009 hasta el cuarto cuatrimestre de 2022.



Fuente Statista

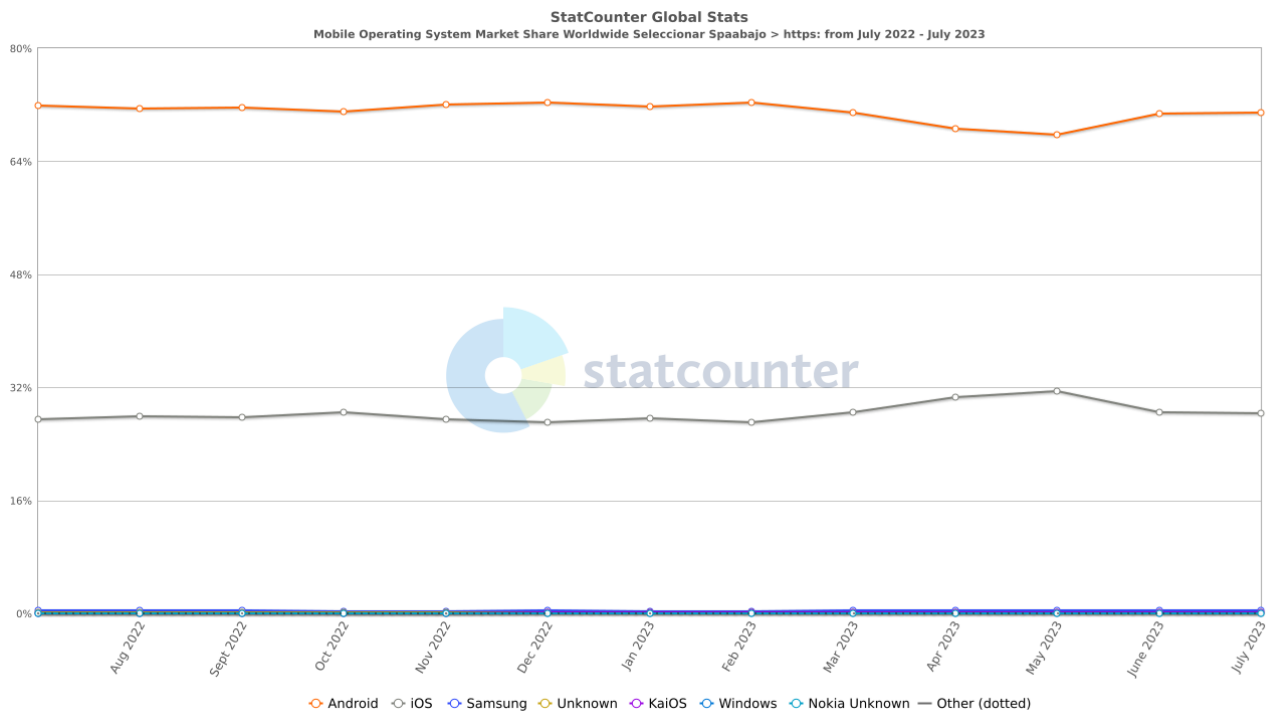
Es evidente que para finales del año 2022, Android había consolidado su posición como líder del mercado, ostentando una participación del 70%. Esta tendencia se manifiesta como una constante a lo largo del tiempo, reflejando la supremacía ininterrumpida de Android en este contexto. Entre los diversos sistemas operativos, apenas destaca un único competidor digno de mención: iOS, que acapara prácticamente el resto del mercado, dejando escaso espacio para otros actores; la presencia de Windows Phone y Samsung es prácticamente marginal y carece de influencia en el panorama global.

• Android	71.8%
• iOS	27.6%
• KaiOS	0.08%
• Windows Phone	0.02%
• Series 40 (Nokia)*	0.01%
• Symbian OS*	0%
• Samsung	0.34%
• BlackBerry OS	0%
• Unknown / Other	0.15%

incluso algunos lenguajes que fueron importantes en su momento como Blackberry o Symbian han desaparecido casi completamente.

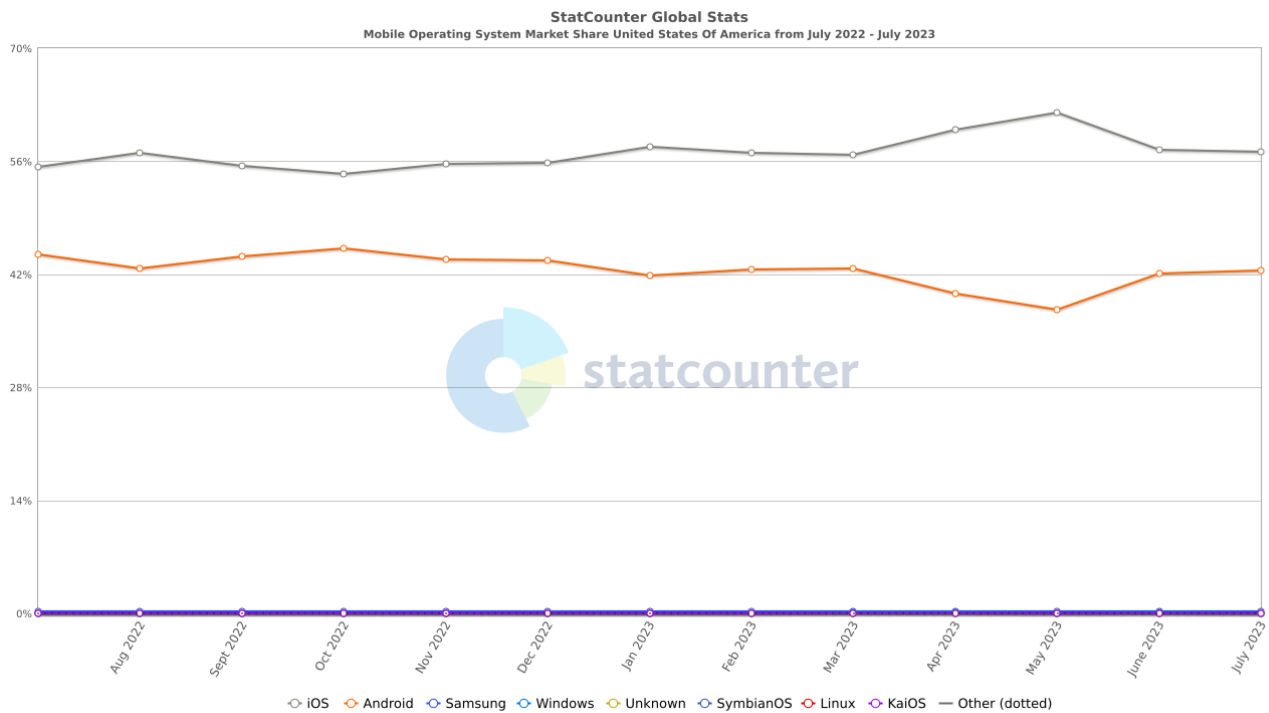
### Mercado Nacional

En el enlace proporcionado en esta página, se ofrece una gráfica del mercado en España, la cual refleja una situación similar a la a nivel global: Android ostenta un dominio del 80%, mientras que iOS representa un 20% de cuota.



### Mercado EE.UU.

Sin embargo, es importante mencionar que estos porcentajes pueden variar en distintas regiones del mundo. Tomemos, por ejemplo, el análisis de la gráfica en EE. UU., donde esta distribución es más equilibrada, con un 56% para Android y un 42% para iOS.



## Mercado

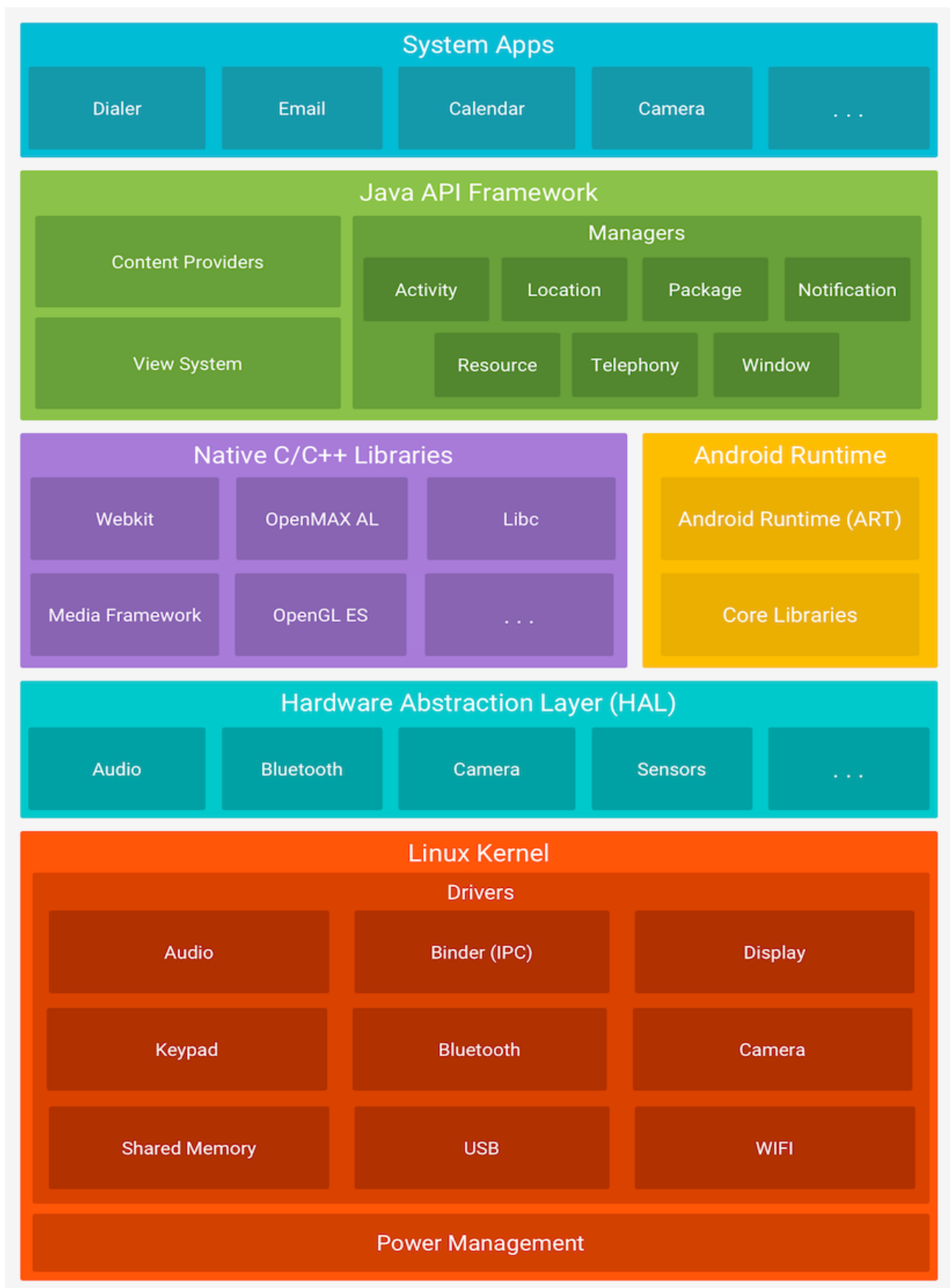
En esta página puedes investigar cómo se distribuye el mercado en más países.

## Arquitectura Android

Android es un sistema operativo meticulosamente diseñado para funcionar en una amplia gama de dispositivos móviles, especialmente aquellos equipados con pantallas táctiles. Esta plataforma se caracteriza por su versatilidad y adaptabilidad a diversas configuraciones de hardware y tamaños de pantalla.

Entre las ventajas más destacadas de Android se encuentra su capacidad para operar en una variedad de dispositivos a pesar de las diferencias en su hardware subyacente. Esta flexibilidad fomenta la innovación, ya que los fabricantes pueden personalizar sus dispositivos mientras se benefician de la estabilidad y consistencia proporcionada por el sistema operativo Android.

En la siguiente imagen podemos ver la arquitectura de la plataforma de Android:



Como podemos ver, en la base de esta estructura reside un núcleo de Linux, similar al de nuestros ordenadores. Sin embargo, lo que varía son las capas superiores que lo complementan. Linux se encargará de la gestión a bajo nivel, administrando el hardware, los procesos, la memoria, etc. de manera eficiente y de forma transparente para el usuario.

Sobre este núcleo, encontramos una capa de abstracción de hardware conocida como “Hardware Abstraction Layer” (HAL). Esta capa de software proporciona una abstracción que aísla las aplicaciones y las bibliotecas de las particularidades específicas del hardware subyacente.

Posteriormente, nos encontramos con un conjunto de bibliotecas nativas (Native C/C++ Libraries) compiladas específicamente para cada plataforma de hardware en la que estamos trabajando, cubriendo aspectos como gráficos y otros recursos.

Sin embargo, el enfoque clave radica en el “Android Runtime”: la Máquina Virtual en Android (anteriormente conocida como Dalvik). Es en esta capa donde nuestras aplicaciones se ejecutarán. Dicho de otra manera, a diferencia de la compilación nativa en C, nuestro proceso de desarrollo se asemejará más al de Java. Se compila en “bytecodes”, que no dependen del hardware.

Aunque no es idéntico, el proceso es similar, constando de dos etapas:

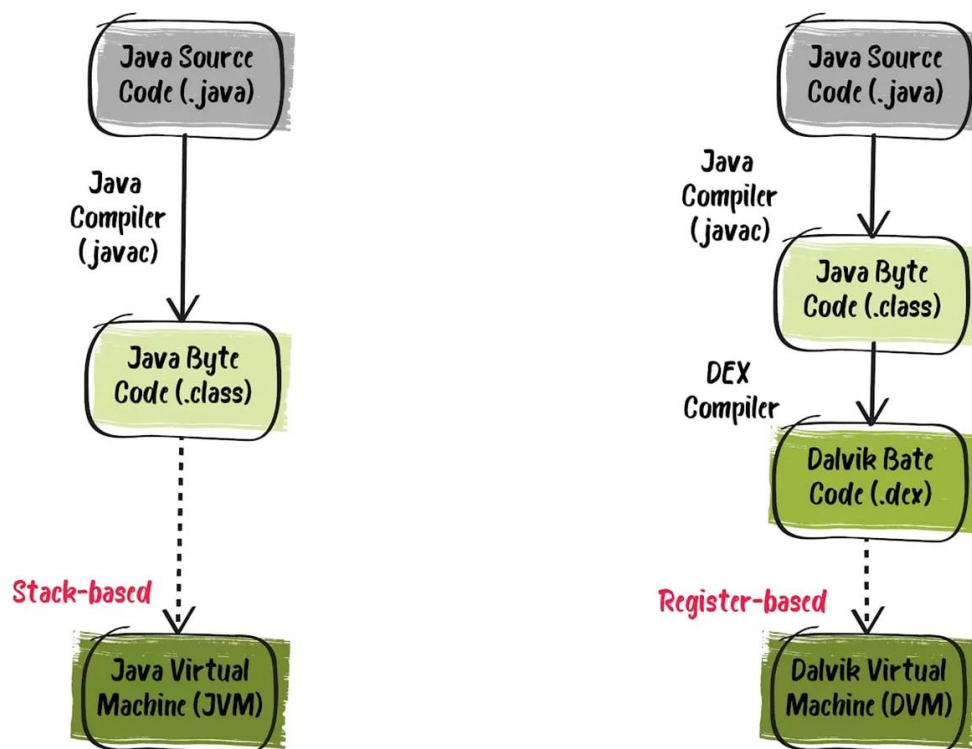
Conversión a bytecodes Java: Inicialmente, el código se convierte en bytecodes Java.

Conversión a bytecodes de Android: Luego, estos bytecodes Java son convertidos a bytecodes específicos de Android.

Este enfoque garantiza una mayor portabilidad y consistencia, ya que el código puede ejecutarse de manera uniforme en diferentes dispositivos, sin depender de las particularidades del hardware subyacente.

## JVM vs ART

En la siguiente imagen podemos ver la diferencia entre la compilación en Java y en Android.



## JVM vs DVM

En Java, generamos nuestro bytecode mediante el compilador, que será interpretado por la máquina virtual de Java. En el contexto de las aplicaciones para Android, encontramos una primera fase de compilación similar. No obstante, diverge con una segunda fase de compilación. En la imagen se observa Dalvik, el precursor de ART actual que se introdujo a partir de la versión 4.4 y se estableció como estándar en la versión 5.0.

La diferencia entre Dalvik y ART radica en que con Dalvik, se realizaba una compilación “just in time”, mientras que con la implementación actual (ART), la segunda fase de compilación ocurre durante la instalación de la aplicación. Este cambio se traduce en mejoras en los tiempos de ejecución.

Es crucial entender que en el contexto de Android, no estamos generando un código ejecutable específico para un hardware particular, al igual que en Java. La gran ventaja es que nuestras aplicaciones pueden ejecutarse en cualquier dispositivo sin necesidad de realizar ajustes, siempre y cuando dispongamos de la máquina virtual correspondiente.

Última actualización: 26.09.2023

Android Studio

Para embarcarnos en el desarrollo en Android, como con cualquier otro lenguaje, se requieren algunos elementos clave:

Un editor de texto.

El JDK (Kit de Desarrollo de Java). Hay que hacer compilación a bytecode.

Librerías adecuadas.

Un emulador para ejecutar las aplicaciones y probarlas.

Aunque estos componentes pueden ser instalados por separado, Google nos ofrece una solución más completa: Android Studio, un IDE basado en la plataforma JetBrains. Este IDE ya incorpora todo lo necesario para el desarrollo de aplicaciones Android, proporcionando un enfoque integral y eficiente para crear, diseñar y probar aplicaciones en esta plataforma.

Última actualización: 26.09.2023

Subsecciones de Android Studio

Requisitos e Instalación

Requisitos

Puedes encontrar los requisitos para la instalación en el siguiente enlace. Es importante que tengas en cuenta que necesitarás:

Una distribución de 64 bits, que es la norma en la mayoría de los sistemas en la actualidad.

Para una experiencia fluida, se recomienda contar con al menos 8GB de RAM o más. Esto se debe a que Android Studio es un software que se ejecuta en una Máquina Virtual de Java, no es una aplicación nativa.

Además, asegúrate de disponer de al menos 8GB de espacio en disco. Esto se debe a que la instalación no solo incluye el IDE en sí, sino también el SDK de Android, las librerías, herramientas, etc. así como el emulador que permite lanzar dispositivos móviles virtuales para pruebas.

Estos requisitos aseguran un rendimiento y funcionamiento óptimos al utilizar Android Studio.

Instalación

La instalación de Android Studio es un proceso relativamente sencillo. Aquí te proporciono los pasos generales para instalarlo en Windows, macOS y Linux:



Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	<a href="#">android-studio-2022.3.1.19-windows.exe</a> Recommended	1.1 GB	d9f21be76024a40395ed2c27baa4347e1765facf3c46294de69bc8452fe04f17
Windows (64-bit)	<a href="#">android-studio-2022.3.1.19-windows.zip</a> No .exe installer	1.1 GB	dcd93092a4ceb8d7945fc24a48bcba5845df208a597f9f49bd9f0aa5e7a5a8ea
Mac (64-bit)	<a href="#">android-studio-2022.3.1.19-mac.dmg</a>	1.2 GB	2ddcfdd1140a2fa444b089ab48963a28ed357ab0b42087b447bab3a98858d85e
Mac (64-bit, ARM)	<a href="#">android-studio-2022.3.1.19-mac_arm.dmg</a>	1.2 GB	547575e9358a4683133ecf38fbff2128e5e1d2aa8462b46cf662ddc3d90e241d
Linux (64-bit)	<a href="#">android-studio-2022.3.1.19-linux.tar.gz</a>	1.2 GB	250625dcab183e0c68ebf12ef8a522af7369527d76f1efc704f93c05b02ffa9e
ChromeOS	<a href="#">android-studio-2022.3.1.19-cros.deb</a>	933.9 MB	3d26ef5f0e14ffdbb144a5527055e254bfa10ae7624b800a7c7ccb184bd10a06

### En Windows

Descarga el instalador de Android Studio desde el sitio oficial: [Descargar Android Studio](#).

Ejecuta el archivo de instalación descargado.

Sigue las instrucciones del asistente de instalación. Puedes elegir la ubicación de instalación y otras configuraciones.

Una vez instalado, inicia Android Studio.

### En macOS

Descarga el archivo de instalación de Android Studio desde el sitio oficial.

Arrastra el ícono de Android Studio a la carpeta de “Aplicaciones” para instalarlo.

Abre “Aplicaciones” y ejecuta Android Studio.

Puedes ser redirigido a una ventana para instalar el JDK si no lo tienes ya instalado. Sigue las instrucciones en pantalla.

### En Linux

Descarga el archivo de instalación de Android Studio desde el sitio oficial.

Extrae el archivo descargado en la ubicación que prefieras.

En la carpeta extraída, ejecuta el archivo studio.sh desde la terminal para iniciar Android Studio.

Puedes ser redirigido a una ventana para instalar el JDK si no lo tienes ya instalado. Sigue las instrucciones en pantalla.

Una vez que hayas instalado Android Studio, es posible que debas configurar algunos ajustes iniciales, como la descarga de componentes adicionales y configuración del emulador. Ten en cuenta que el proceso exacto puede variar según las versiones y actualizaciones del software.

Siempre es recomendable consultar la documentación oficial de Android Studio para obtener instrucciones actualizadas y detalladas según tu sistema operativo.

Última actualización: 17.01.2024

### Configuración IES

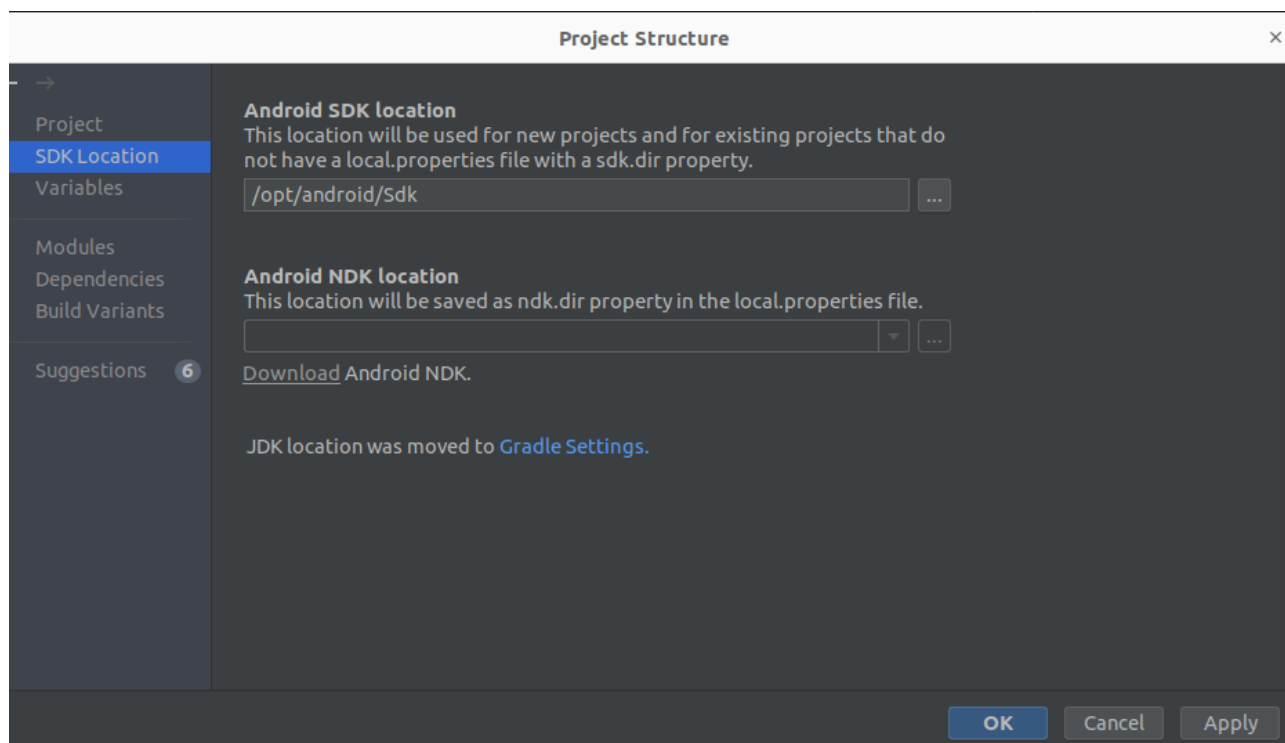
Vamos a trabajar en el Sistema Operativo Debian.

### SDK

Para liberar espacio en disco relacionado con nuestro perfil de usuario, debemos comprobar la ubicación de nuestro SDK y obtenerlo desde la ruta “/opt/android/SDK”. Para hacerlo, siga estos pasos:

Diríjase a “File” -> “Project Structure” -> “SDK Location”.

En la sección correspondiente, cambie la ubicación actual del SDK a “/opt/android/SDK” o “D:\MaquinasVirtuais\sabela\Android” en Windows.  
Si ya había creado un SDK en su espacio de usuario previamente, elimínelo.



### Guardar proyectos

Los proyectos deben estar guardados en la ruta “/opt/android/Projects”. Crear la carpeta si es necesario.

### Actualización de Android Studio

Para llevar a cabo la actualización de Android Studio en el entorno del IES, siga estos pasos:

Acceda a la página oficial de Android Studio y descargue la versión más reciente. Esto descargará una carpeta comprimida.

Asegúrese de que Android Studio está cerrado y, a continuación, descomprima la carpeta descargada.

Copie la carpeta descomprimida llamada “AndroidStudio” en la ubicación /opt/android, reemplazando cualquier versión anterior que pueda existir en ese directorio. Asegúrese de que la carpeta anterior llamada “android-studio” en esa ubicación haya sido eliminada previamente. Finalmente, elimine de forma permanente las carpetas que ya no necesite, incluso desde la papelera. Este proceso garantizará una actualización exitosa de Android Studio en el entorno del IES.

Autor/a: Sabela Sobrino Última actualización: 06.11.2024

### Primer Proyecto

#### Crear un proyecto

Para crear un proyecto en Android Studio debemos seguir los siguientes pasos:

Abre Android Studio y selecciona “New Project”.

Selecciona la plantilla “Empty Activity”.

Detalles de configuración:

Asigna un nombre al proyecto, como “UF1\_UD01\_1\_Hola Mundo”.

Se creará una carpeta para el proyecto en tu directorio seleccionado.

El lenguaje predeterminado suele ser Kotlin, pero también puedes elegir Java.

Define la versión mínima de la API de Android que usará tu aplicación. Un valor más alto te da acceso a las últimas características, pero puede limitar la compatibilidad con dispositivos más antiguos.

Haz clic en “Finish” para completar la configuración inicial.

Android Studio descargará y configurará los elementos necesarios para tu proyecto.

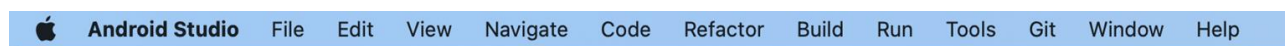
Una vez finalizado, verás una vista familiar similar a otros entornos de desarrollo integrados.

Video: [https://youtu.be/kZSO\\_r3WL3Y](https://youtu.be/kZSO_r3WL3Y)

### Interfaz del IDE

La interfaz del IDE de Android Studio está diseñada para brindar a los desarrolladores un entorno de trabajo eficiente y completo para la creación de aplicaciones Android. A continuación, se describen los elementos clave que puedes encontrar en la interfaz de Android Studio:

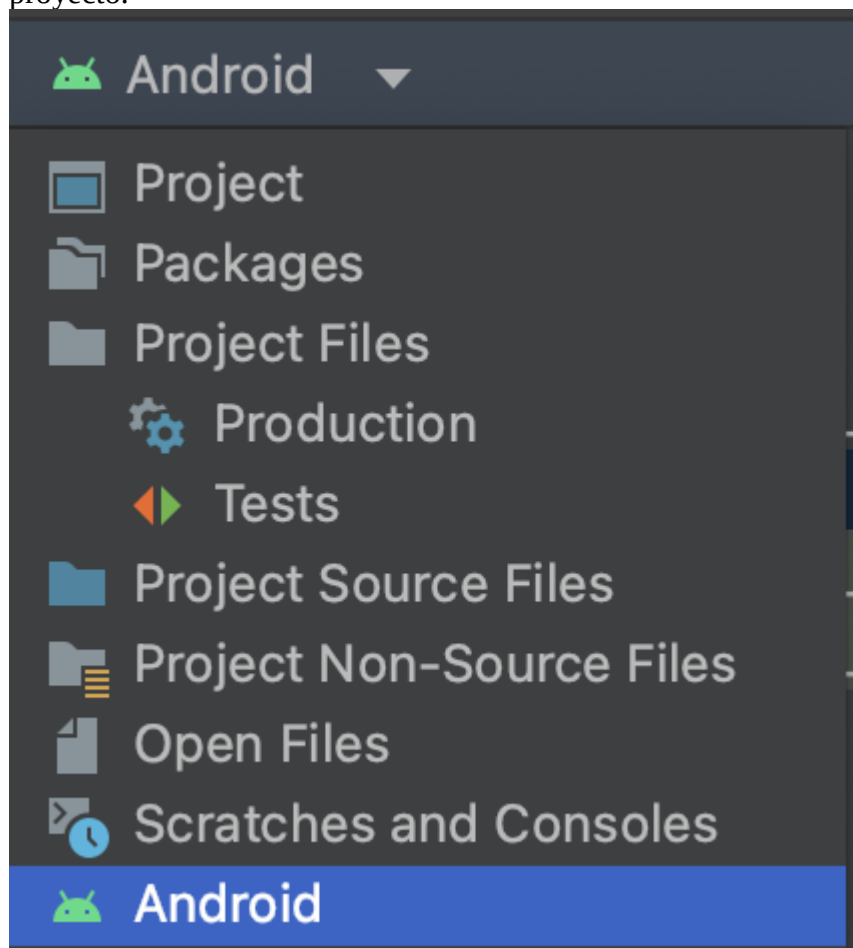
**Barra de Menú:** En la parte superior, encontrarás opciones y menús para realizar acciones, configuraciones y acceder a diversas funcionalidades del IDE.



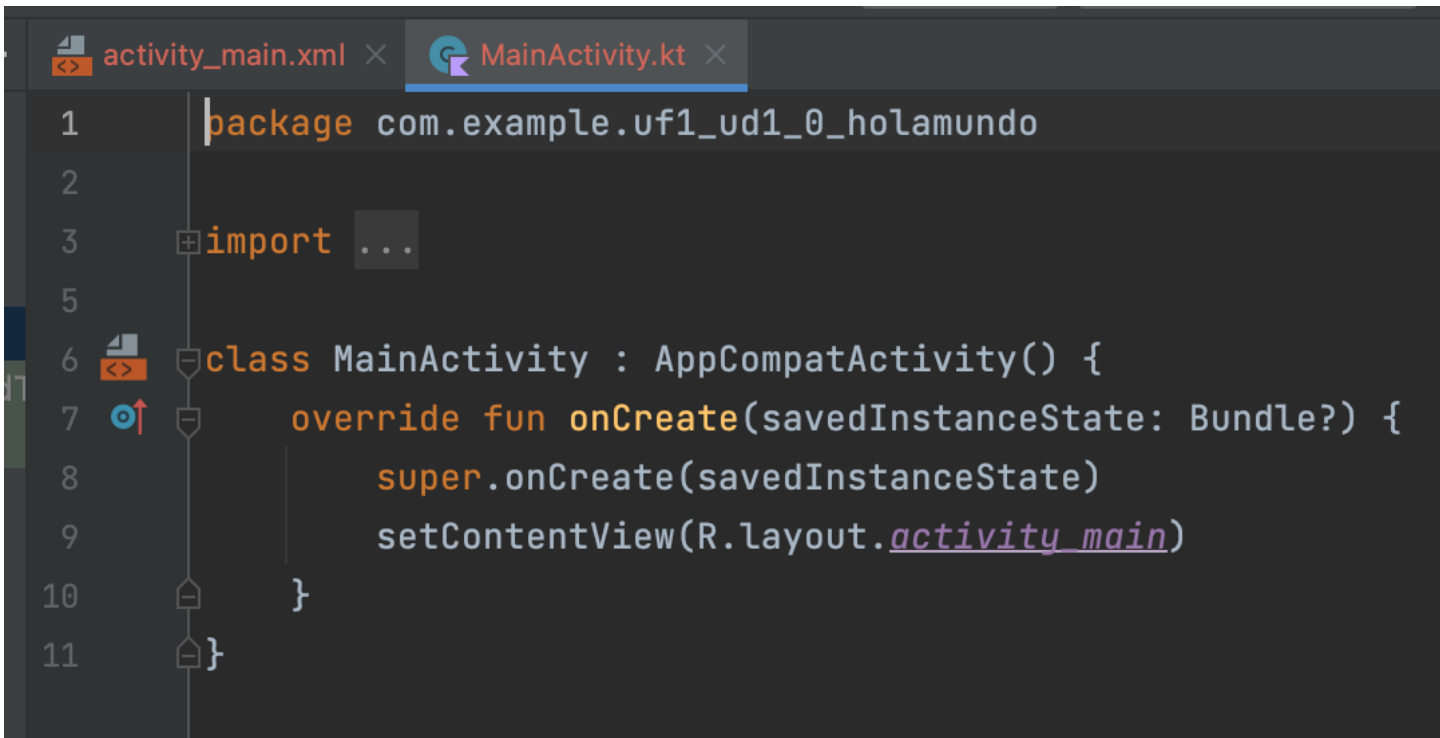
**Barra de Herramientas:** Justo debajo de la barra de menú, se encuentran iconos de acceso rápido a las funciones más utilizadas, como la compilación, la ejecución y la depuración.



**Barra de Navegación Lateral:** En el lado izquierdo, se encuentra la barra de navegación que incluye el “Project Explorer” (Explorador de Proyectos), donde puedes ver y navegar por la estructura de tu proyecto.

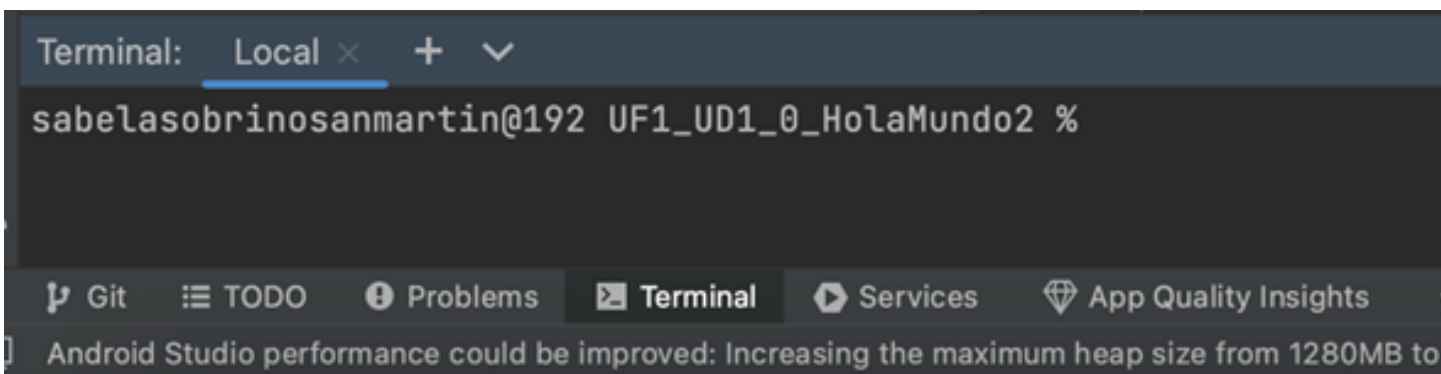


4. **Editor de Código:** La mayor parte del espacio en pantalla se dedica al editor de código. Aquí es donde escribirás y editarás el código fuente de tu aplicación.

A screenshot of the Android Studio IDE showing the MainActivity.kt file. The editor has a dark theme. At the top, there are two tabs: 'activity\_main.xml' and 'MainActivity.kt'. The 'MainActivity.kt' tab is active. The code is as follows:

```
1 package com.example.uf1_ud1_0_holamundo
2
3 import ...
4
5
6 class MainActivity : AppCompatActivity() {
7     override fun onCreate(savedInstanceState: Bundle?) {
8         super.onCreate(savedInstanceState)
9         setContentView(R.layout.activity_main)
10    }
11 }
```

5. **Barra de Estado:** En la parte inferior, encontrarás información relevante sobre el estado actual de tu proyecto, como los errores y advertencias en el código, el estado de la compilación y más. En esta barra también puedes encontrar una terminal integrada que te permite ejecutar comandos de terminal directamente desde el IDE.

A screenshot of the bottom part of the Android Studio IDE. It shows the 'Terminal' tab with the command prompt 'sabelasobrinosanmartin@192 UF1\_UD1\_0\_HolaMundo2 %'. Below the terminal is a toolbar with icons for 'Git', 'TODO', 'Problems', 'Terminal', 'Services', and 'App Quality Insights'. At the very bottom, there is a status bar with the text 'Android Studio performance could be improved: Increasing the maximum heap size from 1280MB to...'

## Ficheros

Cuando creas un proyecto en Android Studio, se generan varios archivos automáticamente para ayudarte a comenzar con el desarrollo de tu aplicación. Dos de estos archivos clave son:

### MainActivity.kt

Este archivo lleva el nombre de “MainActivity.kt” y es un archivo de código fuente escrito en el lenguaje de programación Kotlin. La actividad principal es el punto de entrada de tu aplicación Android. Es el primer lugar que se ejecuta cuando se inicia la aplicación y contiene la lógica para la interfaz de usuario y las interacciones iniciales. En este archivo, encontrarás funciones y métodos que definen cómo se comporta y se ve la pantalla principal de tu aplicación. Aquí es donde manejarás las acciones del usuario y los eventos.

## activity\_main.xml

El archivo `activity_main.xml` es un archivo XML que define la interfaz de usuario de la actividad principal de tu aplicación Android. Este archivo determina cómo se verá y se organizará la pantalla principal de tu aplicación en términos de diseño y elementos visuales. En este archivo puedes definir la disposición de los elementos en la pantalla, como botones, imágenes, campos de texto y otros componentes visuales. Utilizas etiquetas XML para crear y configurar estos elementos.

Puedes usar la vista de diseño en Android Studio para arrastrar y soltar elementos y organizarlos visualmente en la pantalla. El archivo XML se genera automáticamente según las acciones que realices en la vista de diseño.

## Relación con MainActivity

El archivo `activity_main.xml` está vinculado a la `MainActivity.kt` (o `.java`, según el lenguaje que estés utilizando). En el código de la actividad principal, puedes acceder y manipular los elementos de la interfaz definidos en `activity_main.xml`.

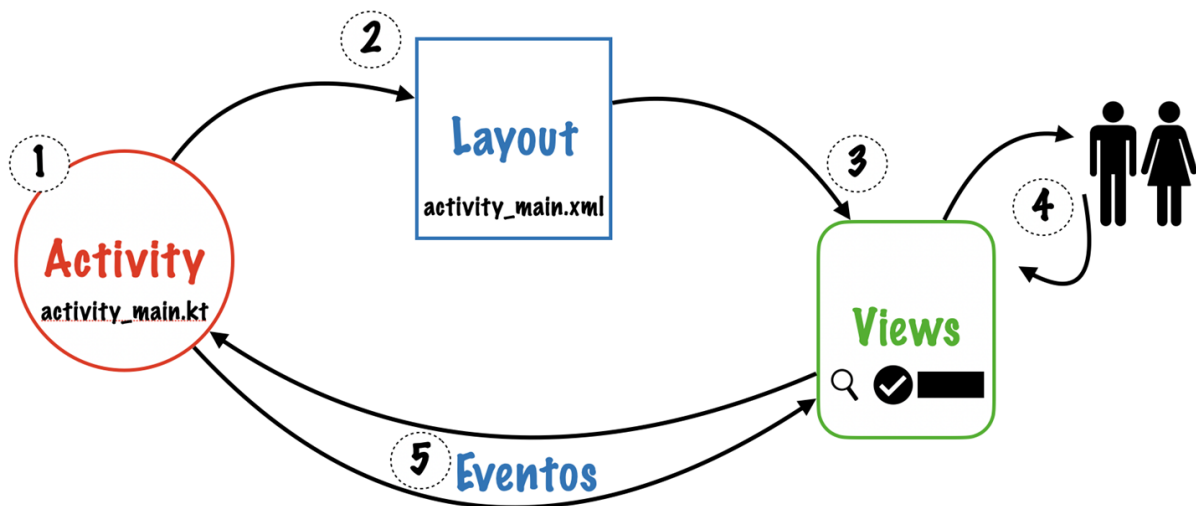
## AndroidManifest.xml

Otro archivo importante, es el archivo que lleva el nombre de “`AndroidManifest.xml`”. Es un archivo XML de configuración crucial para una aplicación Android. El manifiesto es como el “documento de identidad” de tu aplicación. Contiene información esencial que Android necesita para comprender y ejecutar la aplicación correctamente. También proporciona permisos, configuraciones y metadatos. Indica cuál es la actividad con la que se inicia nuestra aplicación:

```
<activity  
    android:name=".MainActivity"
```

## Actividades y Layouts

El siguiente esquema muestra el ciclo de vida de una app Android:



En una app Android el código residirá en una serie de componentes conocidos como actividades. Cuando lanzamos nuestra aplicación, cada actividad se alinea con un conjunto de diseños (XML) que definen la apariencia de las diversas pantallas de nuestra aplicación.

Durante el proceso de lanzamiento, se realiza lo que se denomina “inflado”, que es el acto de construir la representación visual de la pantalla en función de los diseños definidos en los archivos XML correspondientes.

Este proceso da lugar a la generación de una secuencia de eventos, los cuales serán registrados en nuestra actividad. Basados en estos eventos (click, desplegable, etc.), tendremos la capacidad de actualizar la información que está siendo mostrada en esa pantalla particular.

En el transcurso del curso, nuestra atención estará focalizada en dos aspectos principales: las actividades y los diseños. Cuando creamos una aplicación, el proceso genera automáticamente tanto un diseño (basado en la plantilla seleccionada) como una actividad.

### Explorador del Proyecto

Dentro de Android Studio, se presenta una variedad de vistas o perspectivas que permiten a los desarrolladores interactuar y trabajar con distintos aspectos del proceso de creación de aplicaciones.

### Vista “Android”

Una de las perspectivas clave y frecuentemente utilizada es la vista “Android”.

Esta perspectiva Android en Android Studio es donde ocurre la mayor parte del trabajo en el desarrollo de aplicaciones. Proporciona un espacio organizado y específicamente diseñado para crear, diseñar, codificar y depurar aplicaciones Android.

### Vista “Project Files”

Otra de las perspectivas que utilizaremos será la perspectiva “Project Files” (Archivos del Proyecto) en Android Studio es una vista que te permite explorar y gestionar los archivos y carpetas de tu proyecto de manera detallada. Esta perspectiva está diseñada para brindarte un acceso rápido y completo a todos los elementos que componen tu aplicación, desde el código fuente hasta los recursos y archivos de configuración.

### Carpeta “res”

La carpeta “res” (Resources) y la subcarpeta “layout” en Android Studio desempeñan un papel crucial en el desarrollo de aplicaciones Android. Estas estructuras organizativas son fundamentales para separar y gestionar los recursos visuales y de diseño de una aplicación de manera eficiente.

La Carpeta “res” (Resources) almacena todos los recursos utilizados por la aplicación, como imágenes, archivos de diseño, valores de cadena, estilos y más. La separación de recursos del código fuente permite una gestión más eficiente y una mejor organización. Facilita la internacionalización, la adaptación a diferentes dispositivos y versiones de Android, y el mantenimiento en general. Dentro de “res”, encontrarás diversas subcarpetas que almacenan diferentes tipos de recursos, como “drawable” para imágenes, “values” para valores de recursos, “layout” para archivos de diseño, entre otras.

La Subcarpeta “layout” dentro de “res” almacena los archivos de diseño XML que definen la disposición y apariencia de las pantallas de la aplicación. Los archivos de diseño permiten crear interfaces de usuario coherentes y atractivas. Al separar el diseño del código fuente, los diseñadores y desarrolladores pueden colaborar de manera efectiva y los cambios en el diseño no afectan directamente el código funcional.

<https://youtu.be/IGrDv7qZzsk>

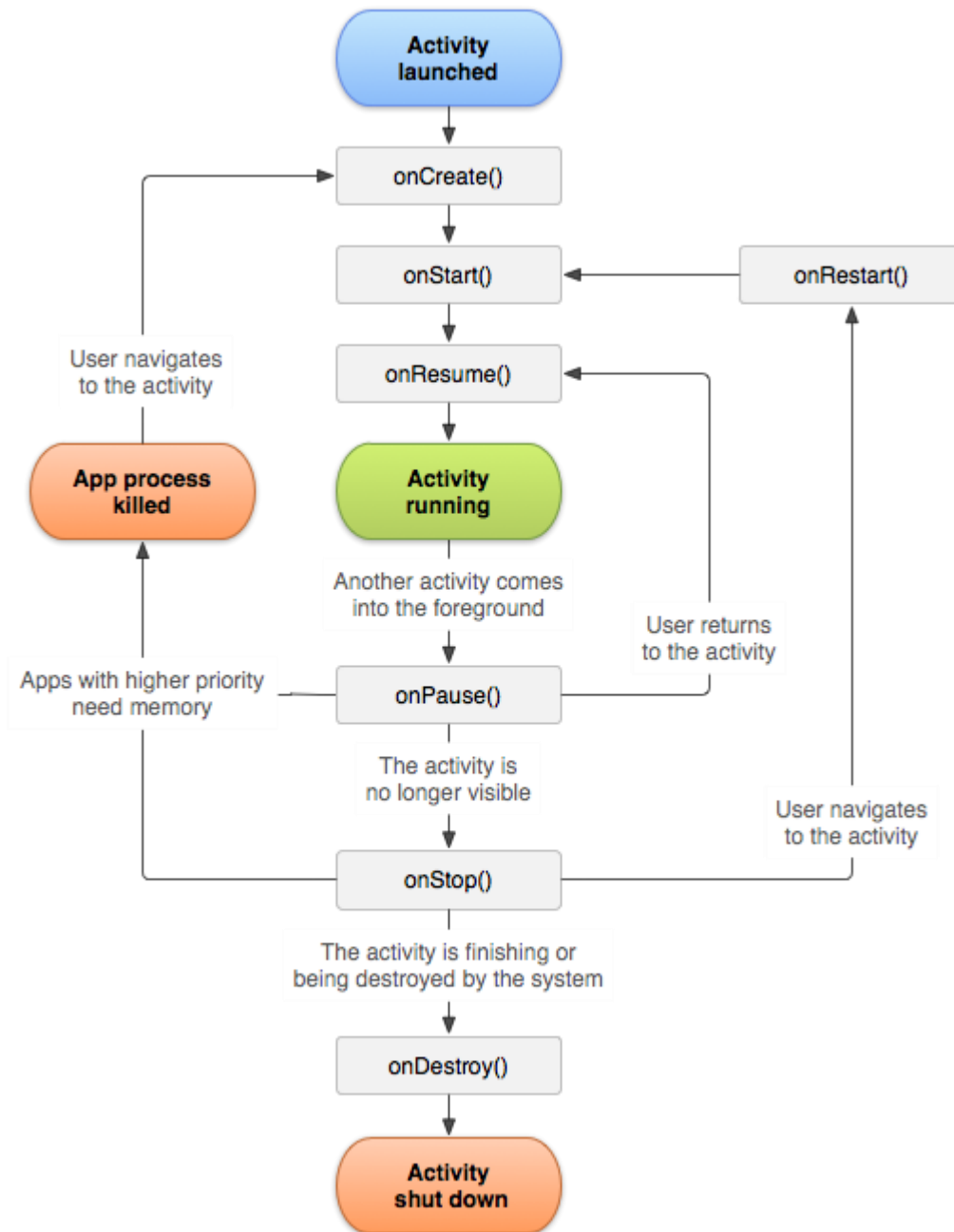
### Main Activity

Las actividades en Android son representadas por clases que heredan de la clase Activity o de alguna de sus subclases. Cuando creamos nuestro proyecto, se está haciendo uso de una de sus subclases llamada “AppCompatActivity”, la cual proporciona un comportamiento básico para la actividad.

```
// Está heredando de la subclase AppCompatActivity
class MainActivity : AppCompatActivity() {
}
```

### Estados

Lo que tendremos que hacer será sobrescribir el método `onCreate`, el cual es invocado en el momento de crear esa actividad. Las actividades en una aplicación Android atraviesan distintos estados (volveremos a ellos más adelante), y cuando se lanza una actividad, se ejecuta el método `onCreate`. En este método, es necesario crear la interfaz visual asociada a dicha actividad.



Este método recibe como parámetro el estado previo de la aplicación, si es que existe. Si se trata del primer lanzamiento, este parámetro tendrá un valor nulo: `savedInstanceState: Bundle?`. En el punto de creación de la actividad, se invoca el siguiente método:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContentView(R.layout.activity_main)
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
        val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
    }
}
  
```



En la llamada a `setContentView(R.layout.activity_main)`, estás proporcionando el diseño visual (archivo XML) que será la base de la pantalla inicial. Este proceso se conoce como “inflado” y es en esta fase que la interfaz visual es construida a partir del diseño definido en el archivo XML.

`enableEdgeToEdge()` configura el diseño de la actividad para utilizar un modo de pantalla completa o “edge-to-edge” (de borde a borde), es decir, donde la interfaz cubre completamente la pantalla.

La función `setOnApplyWindowInsetsListener` ajusta dinámicamente el padding de la vista principal para respetar las barras del sistema (barra de estado, barra de navegación), asegurando que el contenido no quede oculto por estas barras.

## Recursos

Si observamos este método, notaremos que no lo estamos completando con la ruta exacta del archivo XML, sino que en su lugar, proporcionamos un identificador de recurso (R).

```
setContentView(R.layout.activity_main)
```

En este contexto, se crea una clase denominada R, que contiene variables estáticas relacionadas con cada recurso. Esto nos permite acceder a los recursos de manera sencilla y directa. Estos recursos se encuentran dentro de la carpeta “res”, la cual está reservada para recursos y no para código. Así de una forma sencilla podemos acceder a todos los recursos.

## Interfaz de Usuario

Dirigámonos ahora al archivo XML de la pantalla. Aquí, tenemos la flexibilidad de arrastrar elementos y ajustar valores. Este proceso puede ser llevado a cabo de manera visual mediante la interfaz gráfica o en modo de texto.

Dentro de este archivo tenemos tres posibles interfaces: Code, Design, Split:

Dentro de las dos vistas de diseño disponibles:

La vista de diseño técnico (verde) que presenta detalles más técnicos y específicos.

La vista de diseño en blanco nos brinda una aproximación visual o una representación final de la pantalla.

Es posible desactivar una de estas vistas (mediante el icono con dos capas) o elegir el dispositivo en el cual deseamos previsualizar la interfaz.

Autor/a: Sabela Sobrino Última actualización: 17.01.2024

Android Device

## GIF1

Para ejecutar nuestra aplicación necesitaremos un dispositivo sobre el que ejecutar nuestra aplicación. Durante el proceso de compilación, se creará un paquete (.apk) que contendrá el bytecode de nuestro código fuente y aquellos recursos (imágenes, librerías, etc.) y se instalará sobre un dispositivo para posteriormente poder ejecutarlo.

Android Studio ya incluye un emulador para esos dispositivos. Nosotros lo que tendremos que crear es esa máquina virtual, con un hardware específico (por ejemplo; pixel, samsung, etc.).

Para acceder a este emulador, navegamos a “Tools” y seleccionamos “Devices Manager”. Aquí, tenemos la posibilidad de trabajar con dispositivos virtuales y físicos (mediante conexión USB).

Comencemos con el enfoque en el dispositivo virtual. Si no tenemos uno preexistente, podemos hacer clic en “Create Device”. A continuación, seleccionamos el dispositivo deseado, como un smartphone Pixel 5. Esto nos permite emular el hardware específico de ese dispositivo.

Una vez hecho esto, pasamos a la siguiente etapa. Aquí, seleccionamos el sistema operativo adecuado para el dispositivo virtual. Asegurémonos de que sea una versión de API superior a la que se encuentra en nuestro proyecto. En caso necesario, descargamos esta versión, lo cual nos permitirá utilizarla en otros dispositivos en el futuro.

Damos un nombre al dispositivo y configuramos su modo de inicio.

## GIF2

Para ejecutar la aplicación, simplemente presionamos el botón “Run”. Observamos una interfaz de Android en funcionamiento en el emulador.

Para probar nuestra aplicación, accedemos al código fuente y seleccionamos el dispositivo de destino. De momento, veremos en la pantalla la simple frase “Hello World”, ya que en la plantilla hemos incluido la línea:

```
android:text="Hello World!"
```

Esta frase también se reflejará en nuestro dispositivo móvil, donde la aplicación se instalará como cualquier otra.

GIF3

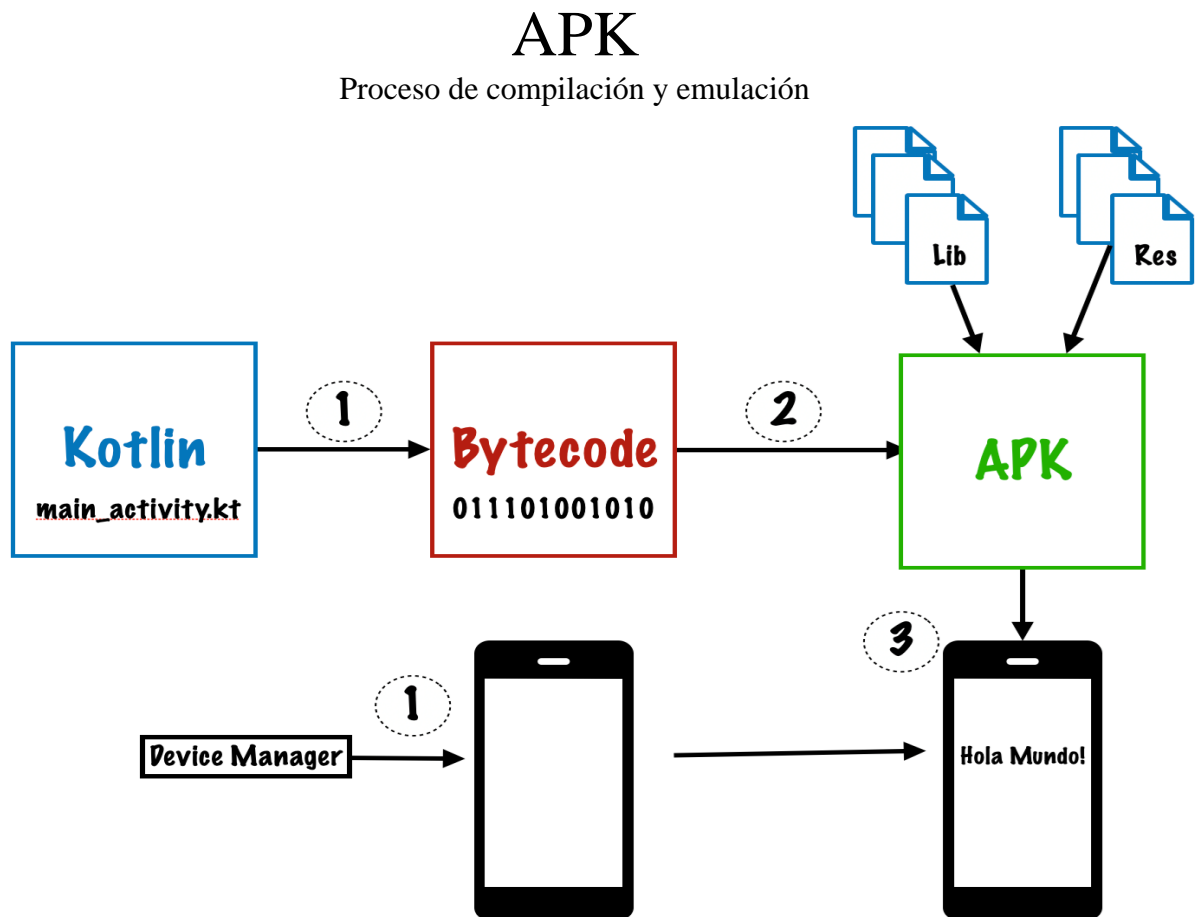
Si deseamos cambiar el texto por “Hola Mundo!”:

```
android:text="@string/hola_mundo"
```

Así, podremos observar cómo el cambio se refleja en el dispositivo móvil.

*Literales*

*Debemos tener en cuenta que el cambio debe realizarse utilizando recursos y no literales directos.  
Ver más en Literales.*



Como podemos ver, a partir de nuestros archivos en Kotlin, se generará una compilación que dará lugar a un código binario. Como vimos, este binario no será nativo, sino ByteCode, el cual se ejecutará en la máquina virtual ART.

Además, nuestra aplicación puede hacer uso de ciertas librerías y diferentes archivos de recursos, que junto con todo esto, contribuirán a la construcción de un conjunto completo, lo que denominamos APK.

Por otro lado, desde el administrador de dispositivos, emulará un dispositivo Android en el que se instalará esta conjunto. Una vez instalado, la aplicación se ejecutará automáticamente.

**Dispositivos Físicos**

Si bien Android Studio nos ofrece la capacidad de emular una variedad de dispositivos virtuales, es igualmente posible incorporar dispositivos físicos en nuestra metodología para obtener una comprensión mejor de nuestra aplicación.

La instalación de estos dispositivos depende tanto del sistema operativo de la máquina hospedera como del software del dispositivo móvil en cuestión. En la siguiente página se encuentra disponible una guía detallada que aborda el proceso de instalación de manera exhaustiva.

<https://developer.android.com/codelabs/basic-android-kotlin-compose-first-app?hl=es-419#0>

## Pimer Proyecto

Vamos a generar una primera aplicación sencilla la cual proporcionará una lista desplegable de variedades de vino. Además, se presentarán algunos ejemplos en función de la elección de la variedad de vino:

### GIF4

Lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

Abrir Android Studio: Abre Android Studio en tu ordenador.

Crear un Nuevo Proyecto:

Selecciona “Start a new Android Studio project” en la pantalla de inicio.

Elige “Phone and Tablet” como tipo de dispositivo.

Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.

Configuración del Proyecto:

En la siguiente pantalla, completa la información básica sobre el proyecto:

Name: Ingresa “UF1\_UD1\_1\_OneMoreWine”.

Package name: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.

Save location: Elige la ubicación donde desees guardar el proyecto en tu sistema.

Language: Selecciona “Kotlin” como lenguaje de programación.

Minimum API level: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.

Finalizar Configuración:

Revisa la configuración y ajusta cualquier otra opción según tus preferencias.

Haz clic en “Finish” para crear el proyecto.

Siguiendo estos pasos, se generará automáticamente una estructura que incluye un archivo MainActivity y un archivo XML en el que podrás desarrollar la interfaz de usuario (activity\_main.xml).

Adicionalmente, se trabajará con otro archivo XML llamado "string.xml" para gestionar la generación de cadenas de texto.

## Diseño de la aplicación

Dentro de la plantilla de nuestra aplicación, es decir dentro del archivo activity\_main.xml, se crean por defecto unos contenedores destinados a alojar nuestros elementos de control, como por ejemplo; casillas de texto, botones y casillas de verificación.

Cada uno de estos componentes, herederos de la clase “View”, debe ubicarse dentro de un contenedor, en este caso denominado “constraintLayout”.

Cada tipo de diseño ("Layout") establece una manera específica de organizar los elementos en la pantalla.

A continuación, se presenta un ejemplo de un diseño (“layout”):

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/main"
tools:context=".MainActivity">
    <!-- Aquí se agregarán los elementos de control -->
</androidx.constraintlayout.widget.ConstraintLayout>
```

## Contenedor

Estos elementos, se crean por defecto. Sin embargo, para nuestro proyecto, no requeriremos un diseño tan complejo. Optaremos por uno más sencillo. Para ello, seguimos los siguientes pasos:

Eliminamos todo el contenido del archivo activity\_main.xml.

A continuación, creamos un nuevo contenedor más simple de tipo `LinearLayout`. Este contenedor es más básico y organiza los elementos en una disposición lineal, ya sea horizontal o verticalmente.:

`<LinearLayout`

```
android:layout_width=""  
android:layout_height=""
```

1. Es importante definir el **espacio XML** donde se especifican todos los atributos y valores. Android Studio autocompletará esta parte al agregar “xmlns” seguido de la URL del sistema.

`<LinearLayout`

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

2. Es necesario definir la **altura** y el **ancho**. Esto es un requisito en el entorno Android. Se pueden utilizar valores numéricos en píxeles, pero también existen constantes como “**match\_parent**” para ajustarse al tamaño del elemento padre, o “**wrap\_content**” para adaptarse al contenido que contenga.

`<LinearLayout`

```
android:layout_height="match_parent"  
android:layout_width="match_parent"
```

En este caso, utilizaremos la constante “**match\_parent**”.

5. Además, es posible especificar si la **disposición** es horizontal o vertical:

`<LinearLayout`

```
android:orientation="vertical">
```

6. Por último, es necesario indicarle un id para que pueda ser llamado desde la actividad:

`<LinearLayout`

```
android:id="@+id/main">
```

En resumen, la estructura de nuestro contenedor **LinearLayout** resultante se ve de esta manera:

`<LinearLayout`

```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:layout_height="match_parent"  
android:layout_width="match_parent"  
android:id="@+id/main"  
android:orientation="vertical">
```

`</LinearLayout>`

Componentes

Lista desplegable

Para generar un lista desplegable, lo primero que hacemos es dirigirnos al menú “Designer” de forma visual, seleccionamos y arrastramos un spinner. Dado que está configurado con orientación vertical, se ubicará en la primera posición disponible, de arriba a abajo. Observamos que se le ha asignado un identificador (que podremos utilizar en el código) de la siguiente manera:

GIF5

<Spinner

```
android:id="@+id/spinner2"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

Podemos cambiar el identificador para facilitar su acceso:

<Spinner

```
android:id="@+id/wineType"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

También notamos que automáticamente se le ha asignado un ancho y una altura. Vamos a modificar el ancho para que se ajuste al contenido que contendrá:

<Spinner

```
android:id="@+id/wineType"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

GIF6

Es importante mencionar que, por ahora, parecerá más pequeño, ya que aún no hemos agregado ningún contenido.

### Centrar Componente

Uso de layout\_gravity:

Podemos utilizar la propiedad android:layout\_gravity para establecer la posición de un componente. Al configurarla como “center,” observamos cómo el componente se desplaza automáticamente al centro del contenedor en el que se encuentra.

Uso de gravity en el propio Layout:

Otra forma de centrar un componente es configurar su posición dentro del propio layout. En el siguiente ejemplo, hemos utilizado un LinearLayout y configurado la propiedad android:gravity como “center.” Esto resulta en la ubicación del componente en el centro de la pantalla:

<LinearLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="match_parent"
```

```

        android:layout_width="match_parent"
        android:orientation="vertical"
        android:gravity="center">

        <Spinner
            android:id="@+id/wineType"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>

```

De esta manera, el componente se centra en toda la pantalla.

Como resumen `layout_gravity` se usa para posicionar un componente dentro de su contenedor, mientras que `gravity` se usa para posicionar todos los componentes secundarios dentro de un contenedor, afectando a la disposición de todos los elementos hijos en relación con el contenido del contenedor en lugar de su contenedor principal.

## Botón

Para incluir un botón justo debajo del spinner, puedes seguir estos pasos:

```

<Spinner
    android:id="@+id/wineType"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button" />

```

Dado que previamente se aplicaron atributos de ancho y altura, ajustaremos nuevamente el ancho del botón para que se adapte al contenido:

```

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />

```

Además, cambiaremos el texto del botón:

```

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Buscar" />

```



## GF7

### Etiqueta de Texto

Si deseas agregar una etiqueta de texto y personalizar su nombre y ancho, puedes hacerlo de la siguiente manera:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Lista de Vinos" />
```

Aquí, hemos definido un elemento TextView con el ID “textView” y hemos ajustado tanto el nombre como el ancho del TextView.

## GIF8

### Apariencia

Vamos a mejorar la apariencia de los elementos introduciendo un espacio entre ellos utilizando la propiedad “padding.” Para lograr esto, configuraremos dos propiedades:

android:gravity="center\_horizontal" para centrar horizontalmente los elementos.

android:padding="16dp" para aplicar un espaciado uniforme alrededor de los elementos.

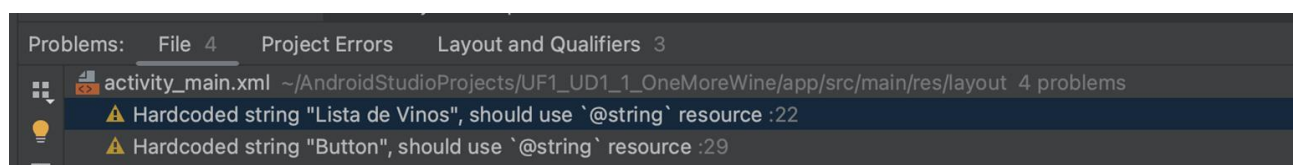
Dentro del diseño, también podemos establecer un margen desde el borde superior. Es importante destacar que “dp” significa densidad de píxeles, lo que nos permite mantener proporciones adecuadas en diferentes dispositivos con tamaños de pantalla variados, como tabletas o teléfonos inteligentes.

Además, podemos utilizar la propiedad “layout\_margin” para separar los elementos entre sí, configurando márgenes derechos e izquierdos según las necesidades de la aplicación. Esto puede variar según la configuración de la aplicación, especialmente si se considera que algunos idiomas, como el árabe, se leen de derecha a izquierda, lo que afectaría la elección entre “margin\_right” y “margin\_start.”

## GIF9

### Recursos

En la pestaña de problemas, observamos la presencia de advertencias que indican la existencia de texto incrustado en el código, en lugar de ser referenciado como un recurso:



Podemos abordar esta cuestión directamente desde estas advertencias. El proceso implica proporcionar el nombre del recurso y su respectivo valor, que luego se almacenará en un archivo ubicado en la carpeta “res”. A continuación, se procederá a reemplazar el texto que anteriormente se encontraba en el código con la siguiente sintaxis:

```
android:text="@string/button_buscar"
```

Este identificador hace referencia a un elemento que se encuentra en la carpeta de recursos dentro de la carpeta “values”:

```
<resources>
    <string name="app_name">UF1_UD1_1_OneMoreWine</string>
    <string name="button_buscar">Buscar</string>
</resources>
```

Este mismo proceso debe repetirse para el resto de las cadenas literales que se encuentren en nuestra aplicación.

GIF10

Localización

La ventaja de utilizar los archivos de recursos radica en la capacidad de localización, lo que permite cambiar el idioma según la configuración del dispositivo. En este caso, vamos a crear una versión en inglés de nuestra aplicación, por lo que necesitamos generar un archivo que contenga traducciones de los textos existentes al inglés.

Para lograr esto, crearemos un nuevo archivo de recursos con el mismo nombre y seleccionaremos el locale correspondiente al inglés, sin especificar una región específica. Al hacerlo, se creará automáticamente una carpeta con el localizador del idioma, que podemos visualizar si cambiamos la vista a “project files”.

GIF11

Inicialmente, el nuevo archivo estará vacío, pero lo completaremos con las mismas cadenas literales que se encuentran en el archivo original, pero con sus valores en inglés:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">OneMoreWine</string>
    <string name="button_buscar">Find</string>
    <string name="text_lista_de_vinos">Wines List</string>
    <string-array name="lista_tipos_vinos">
        <item>White</item>
        <item>Red</item>
        <item>Rose</item>
    </string-array>
</resources>
```

No es necesario realizar ningún otro paso, ya que automáticamente, en función de la configuración del sistema, la aplicación seleccionará el idioma apropiado.

GIF12

Ciclo de Vida

El ciclo de vida de una aplicación Android se refiere a la secuencia de estados y eventos que experimenta una aplicación desde el momento en que se inicia hasta que se cierra. Comprender el ciclo de vida es esencial para desarrollar aplicaciones Android robustas y efectivas.

### Crear proyecto

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro proyecto en Android Studio siguiendo los siguientes pasos:

Abrir Android Studio: Abre Android Studio en tu ordenador.

### Crear un Nuevo Proyecto:

Selecciona “Start a new Android Studio project” en la pantalla de inicio.

Elige “Phone and Tablet” como tipo de dispositivo.

Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.

### Configuración del Proyecto:

En la siguiente pantalla, completa la información básica sobre el proyecto:

Name: Ingresa “UF1\_UD1\_2\_Chronos”.

Package name: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.

Save location: Elige la ubicación donde deseas guardar el proyecto en tu sistema.

Language: Selecciona “Kotlin” como lenguaje de programación.

Minimum API level: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.

### Finalizar Configuración:

Revisa la configuración y ajusta cualquier otra opción según tus preferencias.

Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad MainActivity.kt y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

<https://youtu.be/tpkpoASQhdA>

### Diseño de la aplicación

Lo primero que haremos en nuestro proyecto, será modificar la apariencia de la aplicación. Para lograr esto, nos dirigiremos al archivo `activity_main.xml`, donde actualmente se encuentra un `ConstraintLayout`. Realizaremos el cambio por un `LinearLayout` y definiremos ciertas propiedades:

Utilizaremos autocompletado para el atributo `xmlns`.

Estableceremos el ancho y alto para que se ajusten a las dimensiones del contenedor principal.

Seleccionaremos una orientación vertical.

Aplicaremos un margen de 16dp.

El código resultante quedaría de la siguiente manera:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    >
```

Con estos ajustes, hemos transformado el diseño de `ConstraintLayout` a `LinearLayout` y hemos configurado sus propiedades de ancho, alto, orientación y margen según nuestras necesidades.

Componentes

## Cronómetro

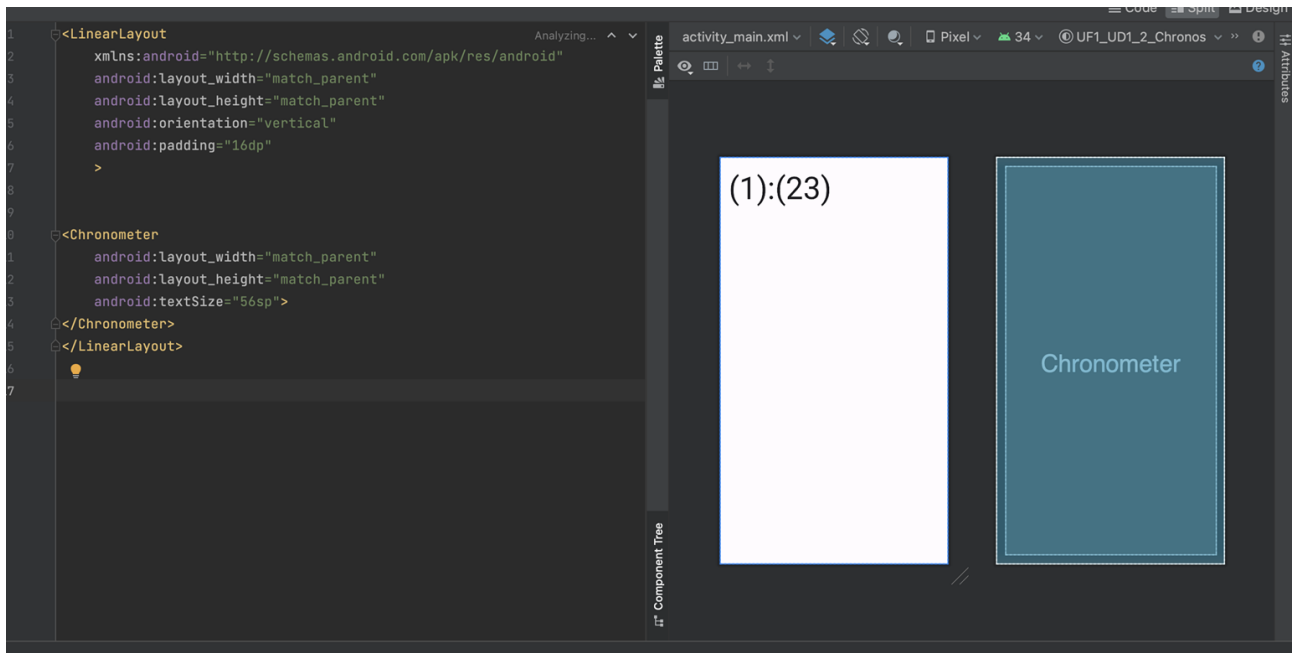
Para añadir un componente de Cronómetro, podemos utilizar el elemento “`Chronometer`”, al que debemos asignar un ancho y un alto. En este caso, se ajustará automáticamente al contenido. Además, es importante asignarle un identificador, que en este caso será “`chr_temporizador`”.

```
<Chronometer
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/chr_temporizador">
</Chronometer>
```

Dado que puede parecer muy pequeño, vamos a ajustar el tamaño de la fuente a 56sp (píxeles escalables, que es la unidad de medida adecuada para fuentes):

```
<Chronometer
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/chr_temporizador"
    android:textSize="56sp">
</Chronometer>
```

Con estos cambios, hemos configurado el componente Chronometer para que se ajuste al contenido y hemos aumentado el tamaño de la fuente a 56sp para hacerlo más legible.



### Centrar elemento

Para centrar el cronómetro en la pantalla debemos modificar el valor de la propiedad gravity del layout:

```
<LinearLayout  
...  
    android:gravity="center_horizontal"
```

### Botones

Para crear los tres botones, utilizaremos el elemento “Button” y configuraremos el alto y el ancho. En este caso, vamos a establecer un ancho fijo para que todos tengan el mismo tamaño (128dp). También añadiremos un margen entre los botones para que haya un espacio de 10dp entre ellos:

```
<Button  
    android:layout_width="128dp"  
    android:layout_height="wrap_content"  
    android:layout_margin="10dp"  
    android:text="@string/start"  
    android:id="@+id/btn_start" />  
  
<Button  
    android:layout_width="128dp"  
    android:layout_height="wrap_content"  
    android:layout_margin="10dp"  
    android:text="@string/pause"  
    android:id="@+id/btn_pause" />
```

```
<Button
    android:layout_width="128dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:text="@string/restart"
    android:id="@+id/btn_restart" />
```

Para agregar texto a cada botón, primero debemos crear tres recursos de tipo String en el archivo “strings.xml” de la carpeta de recursos:

```
<resources>
    <string name="app_name">UF1_UD1_2_Chronos</string>
    <string name="start">Iniciar</string>
    <string name="pause">Pausar</string>
    <string name="restart">Reiniciar</string>
</resources>
```

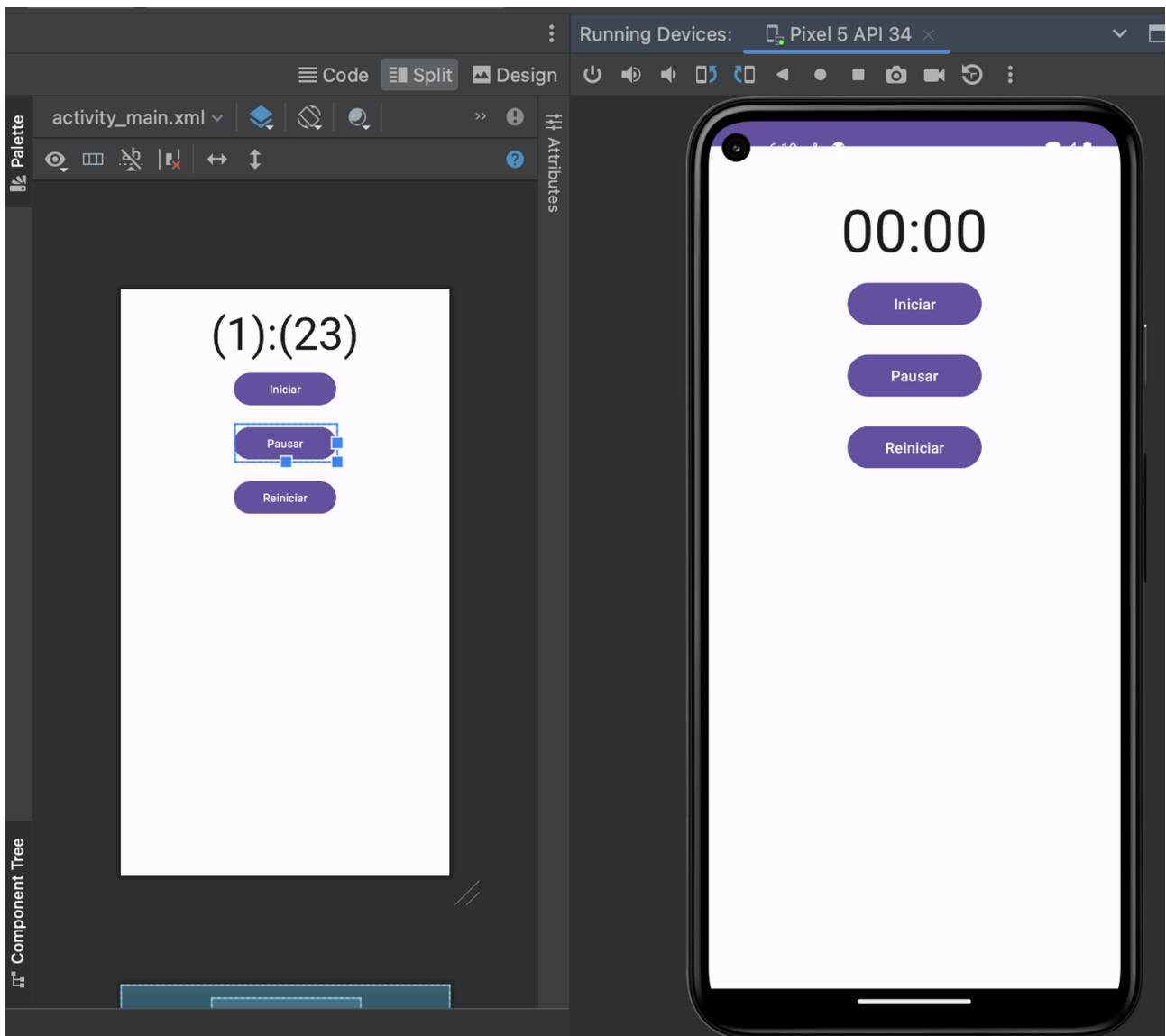
Luego, asignamos el texto a cada botón utilizando la propiedad “android:text” y referenciando los recursos de cadena que creamos anteriormente:

```
<Button
    android:layout_width="128dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:text="@string/start"
    android:id="@+id/btn_start" />

<Button
    android:layout_width="128dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:text="@string/pause"
    android:id="@+id/btn_pause" />

<Button
    android:layout_width="128dp"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:text="@string/restart"
    android:id="@+id/btn_restart" />
```

Una vez agregados estos botones, puedes ejecutar la aplicación para ver cómo se ven en la interfaz de usuario.



### Lógica de los botones

Para implementar la lógica de los botones “Iniciar,” “Pausar” y “Reiniciar,” podemos consultar la documentación oficial de la clase `Chronometer`. En esta clase, encontramos varias operaciones disponibles, incluyendo el acceso a la hora del sistema mediante `SystemClock`. Dos métodos en particular son relevantes para nuestro propósito:

## start

Added in API level 1

```
open fun start(): Unit
```

Start counting up. This does not affect the base as set from `setBase`, just the view display. Chronometer works by regularly scheduling messages to the handler, even when the Widget is not visible. To make sure resource leaks do not occur, the user should make sure that each `start()` call has a reciprocal call to `stop`.

## stop

Added in API level 1

```
open fun stop(): Unit
```

Stop counting up. This does not affect the base as set from `setBase`, just the view display. This stops the messages to the handler, effectively releasing resources that would be held as the chronometer is running, via `start`.

---

Para utilizar estas funciones y desarrollar la lógica de nuestros botones, debemos dirigirnos al archivo Kotlin “MainActivity.kt”.

Variable de Cronómetro

Primero, vamos a crear una variable de tipo Cronómetro en la que almacenaremos nuestro cronómetro y que utilizaremos en varios puntos de nuestro código. Dado que la inicializaremos más adelante, utilizamos la opción “lateinit”. También generaremos otra variable booleana que indicará si el cronómetro está en funcionamiento o no.

```
class MainActivity : AppCompatActivity() {  
    lateinit var chrono: Chronometer  
    var running = false
```

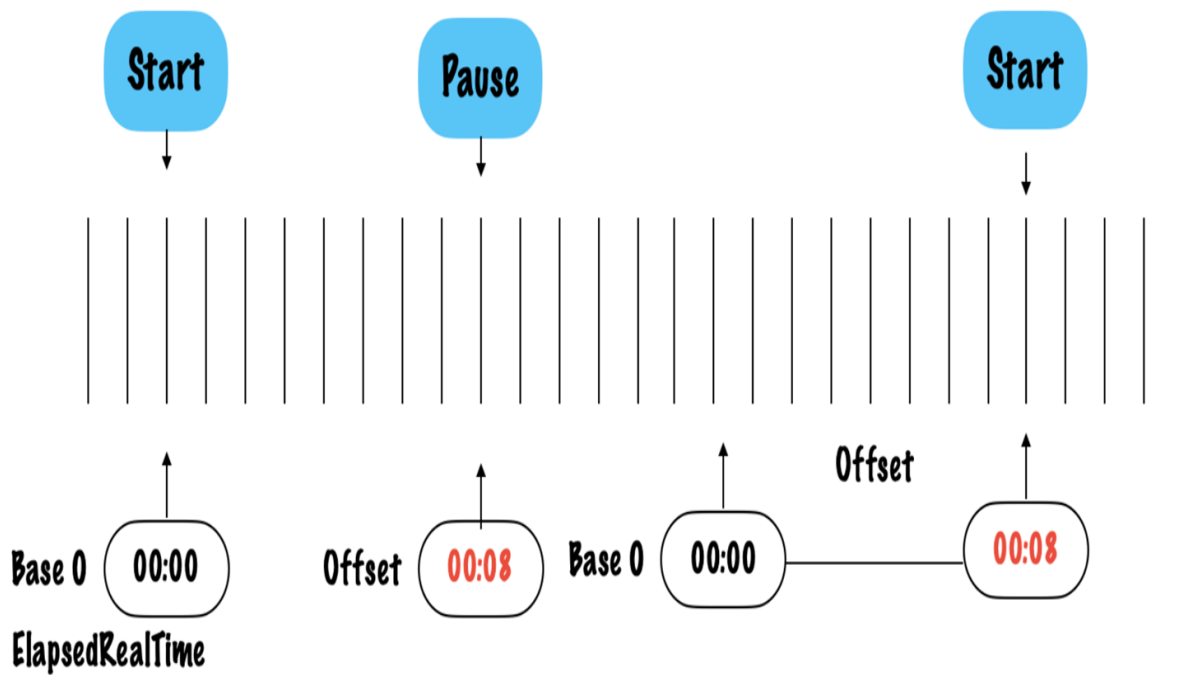
En la función `onCreate`, que se ejecutará cuando se cree la aplicación, recuperaremos la variable Cronómetro con la función `findViewById` y utilizando el ID que habíamos establecido anteriormente.

```
chrono = findViewById<Chronometer>(R.id.chr_temporizador)
```

ElapsedRealTime

En el contexto del Cronómetro, la propiedad `elapsedRealtime` se utiliza para establecer la base de tiempo cuando se inicia el cronómetro o se reinicia. Al establecer la base de tiempo con `SystemClock.elapsedRealtime()`, el cronómetro comenzará a contar desde ese punto en adelante.





Por ejemplo, si deseas medir el tiempo transcurrido desde que se presionó un botón “Iniciar”, puedes hacerlo configurando la base de tiempo del cronómetro en el momento en que se presiona ese botón utilizando `SystemClock.elapsedRealtime()`. Luego, el cronómetro comenzará a contar desde cero, y puedes detenerlo en cualquier momento para obtener el tiempo transcurrido.

#### Botón Iniciar

Para asignar funcionalidad al botón “Iniciar,” primero lo recuperamos y luego utilizamos la función `setOnClickListener` para definir la función lambda que se ejecutará al hacer clic en él:

```
val btnStart = findViewById<Button>(R.id.btn_start)
btnStart.setOnClickListener { }
```

En este caso, verificamos si el cronómetro está detenido (mediante la variable booleana) y establecemos el valor base, es decir, el tiempo a partir del cual comenzará a contar. Una vez configurado, iniciamos el cronómetro y cambiamos el valor de la variable booleana a `true`:

```
btnStart.setOnClickListener {
    if (!running) {
        // Establecer una marca desde la cual comenzará a contar el cronómetro
        chrono.base = SystemClock.elapsedRealtime()
        chrono.start()
        running = true
    }
}
```

#### Botón Pausar

Siguiendo la misma lógica que el botón “Iniciar,” escribimos el código necesario para detener el cronómetro y cambiamos el valor de la variable booleana:

```

val btnPause = findViewById<Button>(R.id.btn_pause)
btnPause.setOnClickListener {
    if (running) {
        chrono.stop()
        running = false
    }
}

```

#### Botón Reiniciar

En este caso, simplemente restablecemos el valor base del cronómetro. No modificamos el estado del cronómetro, por lo que si estaba detenido, seguirá detenido, y si estaba en funcionamiento, continuará en funcionamiento:

```

// Botón Reiniciar
val btnRestart = findViewById<Button>(R.id.btn_restart)
btnRestart.setOnClickListener {
    chrono.base = SystemClock.elapsedRealtime()
}

```

Con estos ajustes, hemos implementado la lógica necesaria para controlar el cronómetro utilizando los botones “Iniciar,” “Pausar” y “Reiniciar.”

Ahora podremos ejecutar nuestra aplicación y comprobar si funciona todo correctamente:

GIF13

#### Problema Pausa

Si profundizamos en el comportamiento de la aplicación, podemos identificar un problema con la función de pausa. Cuando realizamos los siguientes pasos: “**Iniciar - Pausar - Iniciar,**” observamos que el contador se reinicia a 0.

GIF14

Para resolver este problema, calcularemos el **tiempo** transcurrido desde que iniciamos el cronómetro hasta que realizamos una pausa. Cuando se realice una pausa, estableceremos la base del cronómetro en ese punto específico.

Para lograrlo, crearemos una variable llamada offset que almacenará ese valor de **tiempo** transcurrido y la inicializaremos en 0L, ya que los valores de tiempo en el sistema son de tipo Long:

```

var offset = 0L

```

Cuando pulsemos “Pausa,” actualizaremos esa variable con el valor actual del reloj del sistema (elapsedRealtime) y le restaremos el tiempo en el que se encontraba el reloj del sistema en el momento en que iniciamos el cronómetro:

```
offset = SystemClock.elapsedRealtime() - chrono.base
```

Ahora que tenemos el valor de offset definido, cuando hagamos clic en “Iniciar,” restaremos ese valor, teniendo en cuenta que la primera vez este valor será 0L.

Finalmente, al hacer clic en “Reiniciar,” estableceremos offset nuevamente en 0L.

De esta manera, al volver a ejecutar la aplicación, veremos que el problema de la pausa se ha solucionado.

GIF15

Ciclo onCreate

Cuando iniciamos el cronómetro y luego giramos la pantalla, notamos que el temporizador se detiene:

GIF16

Este comportamiento no debería ocurrir y se debe a que cuando giramos el dispositivo Android, la aplicación destruye y vuelve a crear la actividad principal. Esto ocurre porque una aplicación puede tener diferentes actividades para diferentes configuraciones del teléfono, como la orientación de la pantalla o el idioma.

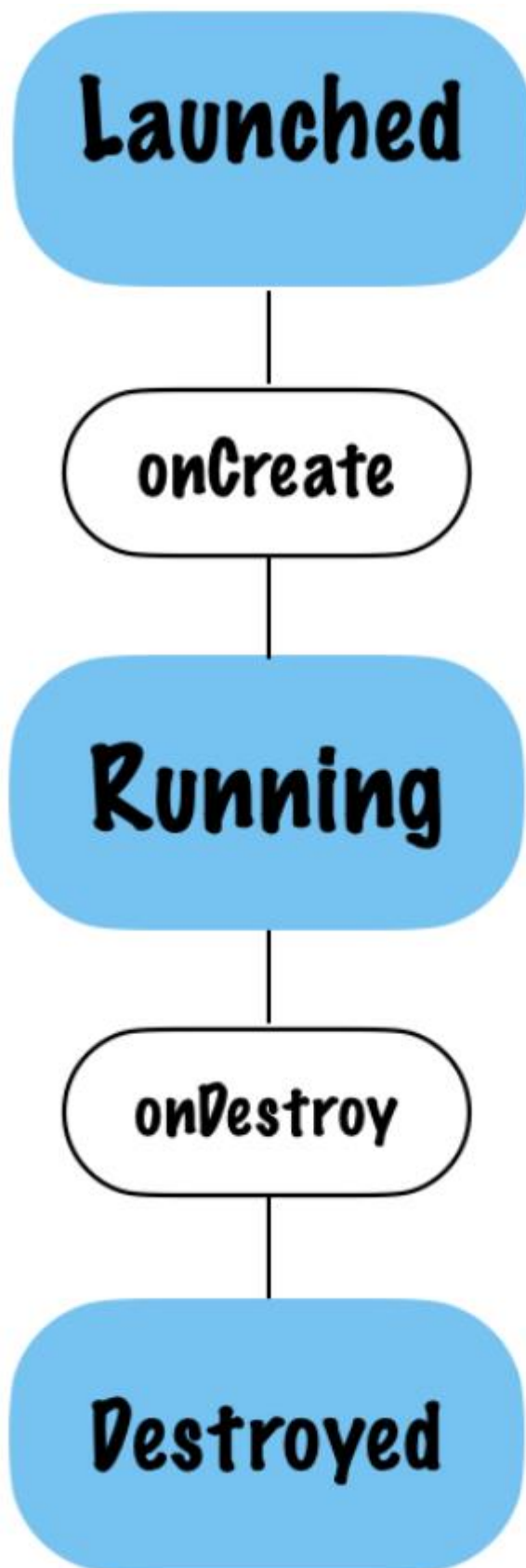
Nuestra actividad principal pasa por varios estados:

**Launched:** Cuando lanzamos la aplicación. En este estado, el objeto está creado y se ejecutará el método onCreate().

**Running:** Luego pasa al estado “en ejecución” (running), donde la actividad se ejecuta mientras es la actividad principal y tiene el foco.

**Destroyed:** Finalmente, pasa al estado destruido (destroyed). En este estado, el objeto deja de existir y se ejecutará el método onDestroy().

Existen más estados intermedios, pero lo relevante en este momento es que podemos acceder a esos métodos que se invocan antes de cambiar de estado y modificarlos, como ya lo hacemos con el método onCreate().



Bundle

Si observamos el método onCreate, vemos que tiene un argumento llamado Bundle. Este es el argumento que debemos utilizar para guardar el estado de nuestra aplicación o cierta información antes de que el método se complete.

Para hacer esto, sobrescribimos el método onSaveInstanceState, que nos permite guardar o recuperar información de ese mapa Bundle.

Podemos guardar las variables running, offset y base. Para ello, definimos unas constantes que nos permitan pasar los valores:

```
class MainActivity : AppCompatActivity() {  
    val RUNNING_KEY = "running"  
    val OFFSET_KEY = "offset"  
    val BASE_KEY = "base"
```

Dentro del método onSaveInstanceState, utilizamos los métodos put para guardar estos valores:

```
override fun onSaveInstanceState(outState: Bundle) {  
    outState.putBoolean(RUNNING_KEY, running)  
    outState.putLong(OFFSET_KEY, offset)  
    outState.putLong(BASE_KEY, chrono.base)  
    super.onSaveInstanceState(outState)  
}
```

De esta manera, tenemos estos datos guardados. Nota que en la última línea, se invoca al método onSaveInstanceState del padre, ya que la actividad también almacena información importante para su funcionamiento interno, que generalmente no es visible para nosotros pero es necesaria para la aplicación.

## Restaurar ese Estado

Si la actividad se tiene que volver a crear por algún motivo, podemos restaurar el estado en la función onCreate. Si el parámetro savedInstanceState no es nulo, recuperamos los valores utilizando el método get:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    chrono = findViewById<Chronometer>(R.id.chr_temporizador)  
  
    if(savedInstanceState!=null){  
        offset = savedInstanceState.getLong(OFFSET_KEY)  
        running = savedInstanceState.getBoolean(RUNNING_KEY)  
    }  
}
```

Si la aplicación estaba en funcionamiento, debemos continuar la ejecución desde el punto en el que estaba:

```
if(running){
    chrono.base = savedInstanceState.getLong(BASE_KEY)
    chrono.start()
}else{
    chrono.base = SystemClock.elapsedRealtime() - offset
}
```

Si el cronómetro estaba funcionando, mantenemos el mismo tiempo (guardado en la variable BASE\_KEY) y llamamos a la función start() para que continúe la ejecución. Si no estaba en funcionamiento, ajustamos la base del cronómetro teniendo en cuenta el valor de offset.

Al ejecutar nuevamente la aplicación, podemos verificar que todo funciona correctamente

Ciclo onStop, onRestart, onStart

Cuando nuestra aplicación pasa a segundo plano debido por ejemplo, a una llamada telefónica o al cambiar a otra aplicación, podemos notar que el temporizador sigue avanzando:

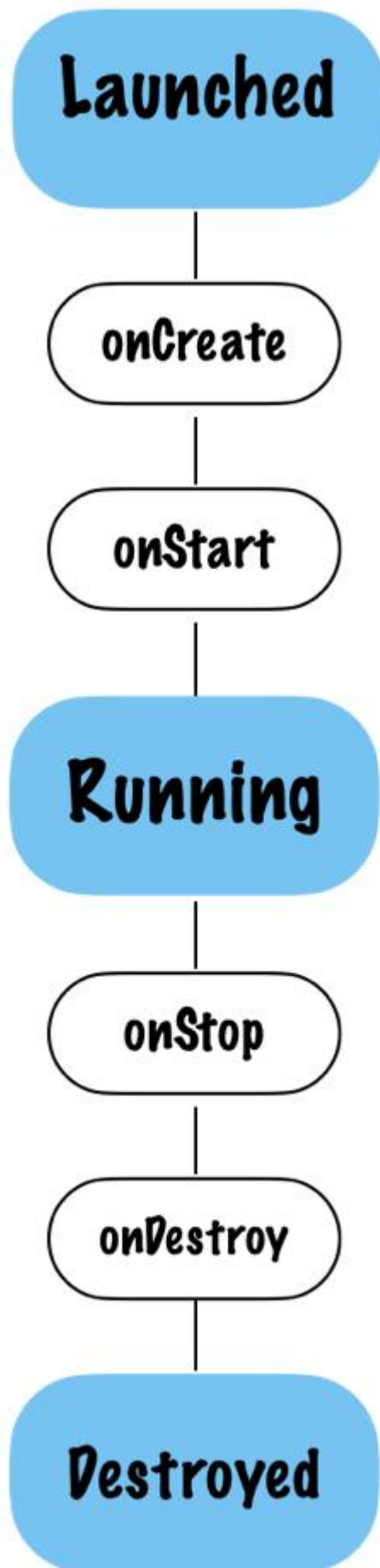
En algunos casos, es deseable que la aplicación solo se ejecute cuando está en primer plano y se detenga cuando está en segundo plano. Para lograr esto, podemos sobrescribir ciertos métodos del ciclo de vida de la actividad.

Siguiendo el esquema de estados que hemos estado utilizando hasta ahora, podemos agregar algunos nuevos:

**Launched:** Cuando lanzamos la aplicación. En este estado, el objeto está creado y se ejecutará el método onCreate(). Después de la ejecución de este método onCreate(), se ejecutará otro método, onStart(), que se invoca cuando nuestra aplicación pasa a primer plano, es decir, cuando se hace visible.

**Running:** A continuación, pasa al estado “en ejecución” (running), donde la actividad se ejecuta mientras es la actividad principal y tiene el foco.

**Destroyed:** Por último, pasa al estado destruido. En este estado, el objeto deja de existir y se ejecutará el método onDestroy(). Antes de ejecutar el método onDestroy(), se ejecutará otro método, onStop(), cuando nuestra aplicación pasa a segundo plano y/o queda oculta, o antes de destruirse. Si la aplicación pasa de segundo plano a primer plano, se ejecutará el método onRestart().



## onStop()

Sobrescribiremos el método onStop() para que si la aplicación pasa a segundo plano, se detenga la ejecución. Sin embargo, debemos almacenar el valor de offset actual para que cuando reanudemos la ejecución, el temporizador continúe desde donde se detuvo (similar a lo que hacemos con el botón de pausa).

```
override fun onStop() {  
    if (running) {  
        offset = SystemClock.elapsedRealtime() - chrono.base  
        chrono.stop()  
    }  
    super.onStop()  
}
```

No cambiamos el valor de la variable booleana running porque cuando la aplicación vuelva a primer plano, la ejecución debe continuar.

## onRestart()

Sobrescribiremos el método onRestart() para que cuando la aplicación vuelva a primer plano, la ejecución se reanude (similar al comportamiento del botón “Iniciar”):

```
override fun onRestart() {  
    if (running) {  
        chrono.base = SystemClock.elapsedRealtime() - offset  
        chrono.start()  
    }  
    super.onRestart()  
}
```

## Ciclo onPause - onResume

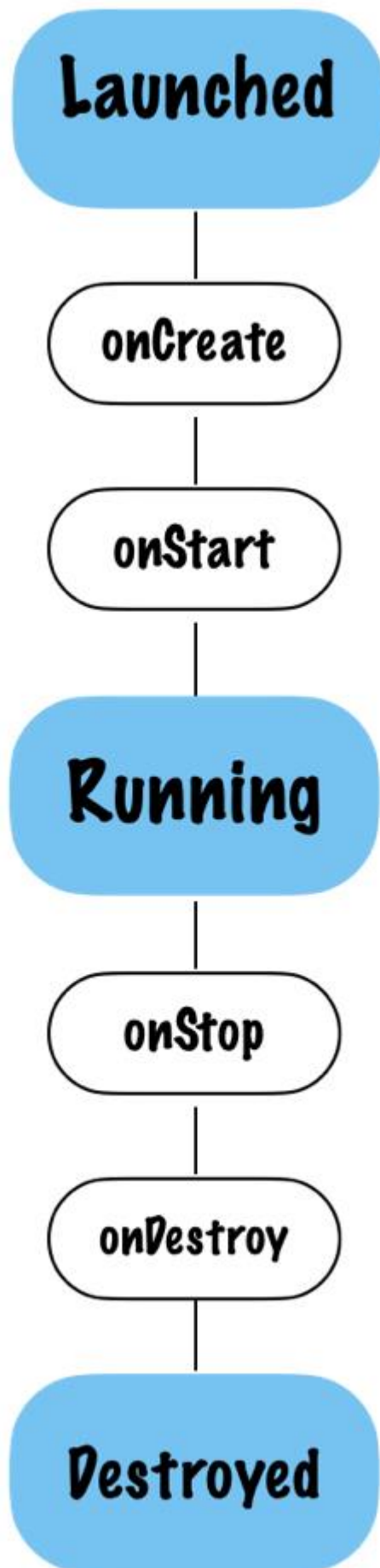
¿Qué sucede si nuestra aplicación continúa en primer plano pero pierde el foco debido por ejemplo, a una notificación de Google u otra interrupción similar? Al completar el esquema de estados de nuestra aplicación, podemos observar que se invocan otros métodos:

**Launched:** Cuando lanzamos la aplicación. En este estado, el objeto está creado y se ejecutará el método onCreate(). Después de la ejecución de este método onCreate(), se ejecutará otro método, onStart(), que se invoca cuando nuestra aplicación pasa a primer plano, es decir, cuando se hace visible.

**Running:** A continuación, pasa al estado “en ejecución” (running), aquí tendremos esa actividad ejecutándose mientras es la actividad principal y tiene el foco. Cuando la aplicación que está en ejecución pierde el foco, se invoca el método onPause(). Si la aplicación recupera el foco, se invocará onResume().



Destroyed: Por último, pasa al estado destruido. En este estado, el objeto deja de existir y se ejecutará el método `onDestroy()`. Antes de ejecutar el método `onDestroy()`, se ejecutará otro método, `onStop()`, cuando nuestra aplicación pasa a segundo plano y/o queda oculta, o antes de destruirse. Si la aplicación pasa de segundo plano a primer plano, se ejecutará el método `onRestart()`.



Para probar el funcionamiento de estos métodos, podemos aprovechar el código de los métodos anteriores y cambiar sus nombres por onPause() y onResume():

```
override fun onPause() {  
    if(running){  
        offset = SystemClock.elapsedRealtime() - chrono.base  
        chrono.stop()  
    }  
    super.onPause()  
}  
  
override fun onResume() {  
    if(running){  
        chrono.base = SystemClock.elapsedRealtime() - offset  
        chrono.start()  
    }  
    super.onResume()  
}
```

El código de la aplicación seguirá siendo el mismo, ya que antes de cambiar a segundo plano, la aplicación perderá el foco.

### Práctica

En este ejercicio, crearemos una aplicación que nos permita cambiar de moneda (euros a dólares) teniendo en cuenta los diferentes estados de la aplicación.

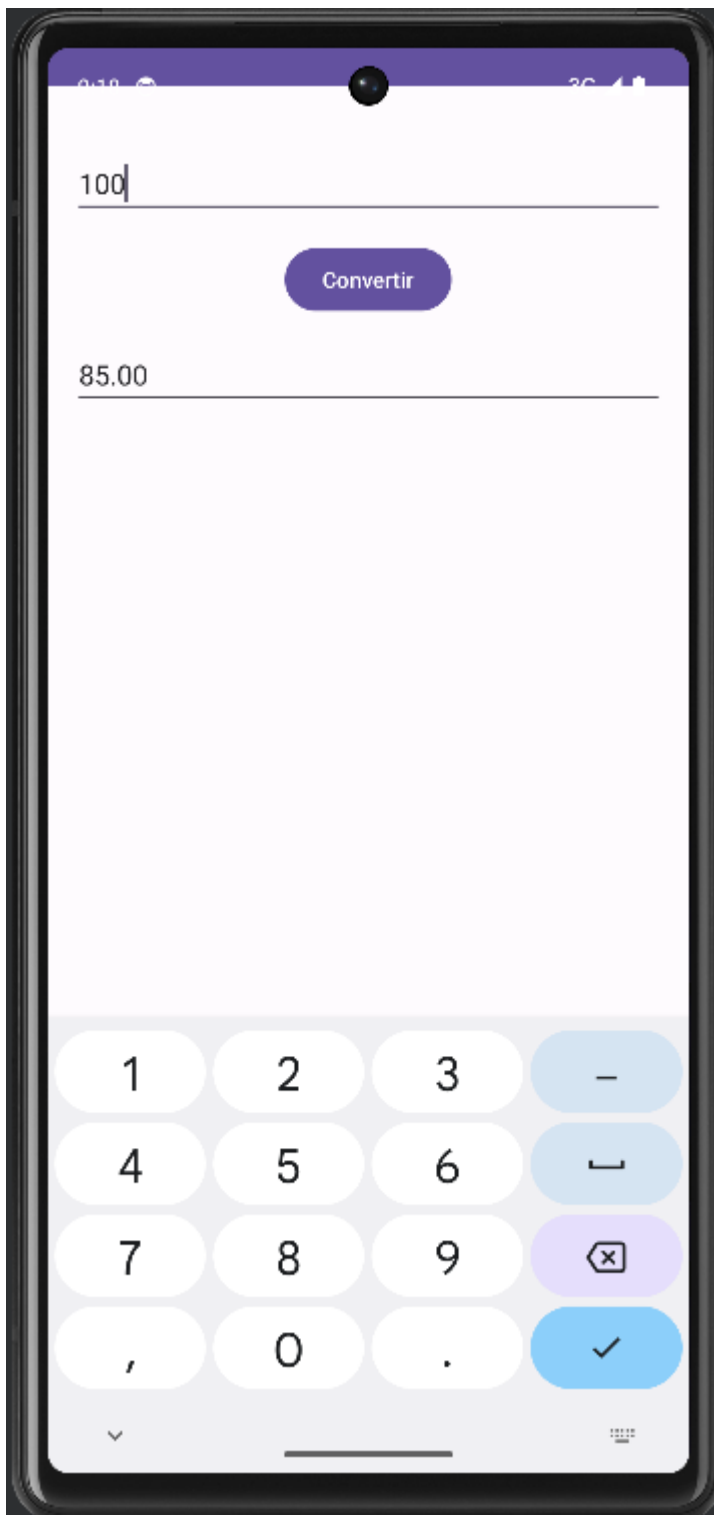
### Instrucciones

Abre Android Studio y crea un nuevo proyecto de Android llamado “UD01\_Practica\_ConversorMoneda”.

Diseña la interfaz de usuario en el archivo activity\_main.xml para incluir los siguientes elementos:

Dos EditText para ingresar la cantidad en dólares y el resultado en euros.

Un Button para iniciar la conversión.



Implementa la lógica para realizar la conversión en la actividad principal (MainActivity.kt) teniendo en cuenta que debe funcionar aunque cambie a segundo plano y/o haya una interrupción.