Programación Multimedia y Dispositivos Móviles

UD 2. Introducción a los dispositivos móviles

Javier Carrasco

Curso 2024 / 2025

Introducción a los dispositivos móviles

2. Introducción a los dispositivos móviles

Se tratará de ofrecer una visión global de la evolución, y situación actual, en lo que respecta a los dispositivos móviles. Además, se establecerá la primera toma de contacto con el entorno de desarrollo (IDE) que se utilizará a lo largo del libro, **Android Studio**, mediante su instalación y creación del primer proyecto (*"Hello World"*). También se abordarán otros conceptos con los que deberás familiarizarte para trabajar con proyectos Android.

2.1. Análisis y limitaciones

En la actualidad se da por sentada la importancia del teléfono móvil, o *Smartphone*, en nuestras vidas, dándole uso tanto para el ocio como para el trabajo. Desde sus inicios, más allá de la aparición del primer **iPhone** en 2007 de la mano de **Apple**, la telefonía móvil ya se planteaba como un reto.

En los años 70, **Motorola** hace una demostración de conexión utilizando una red que se conocería como la red **1G**. En los años 90 empieza a extenderse el concepto de movilidad, apareciendo así la red **2G**. Una década después se da un empujón a las líneas de comunicación y surge la red **3G** y, otros diez años después, aproximadamente, en 2009, se produce la presentación oficial de la red **4G**. Actualmente nos encontramos en plena generación **5G**, de la que se espera llegue a todo el mundo en torno al año 2025.

Los años 90 supusieron el auge de dispositivos como las **PDAs**, pequeños organizadores personales con conectividad móvil, utilizando sistemas operativos como **PalmOS**, **Windows CE** o **BlackBerry OS**. Estos dispositivos, que necesitaban de un lápiz óptico o puntero para funcionar, empiezan a evolucionar, siendo capaces de realizar cada vez más tareas y aumentando sus prestaciones. Fue entonces, en 2007, cuando aparece el primer teléfono móvil, el **iPhone** de **Apple**, dejando a las PDAs como dispositivos obsoletos y, sin la necesidad de utilizar punteros.

En 2008 aparece en escena **Google** con el primer dispositivo con el sistema operativo **Android**. Dicho sistema operativo se lo compró Google a **Andy Rubin** en el año 2005. Mientras tanto, **Microsoft** lanzaba, o más bien evolucionaba Windows CE, su nuevo sistema operativo para dispositivos móviles, **Windows Phone**, enterrando definitivamente a *Symbian*, SO utilizado en aquel momento por Nokia. Estos dos SO, junto con **iOS**, son los tres SO que pelearon por hacerse un hueco en el mercado de la telefonía móvil en los inicios. Actualmente, la batalla ha quedado entre Android y iOS. Según estudios de mercado realizados en 2018, el uso de Android se encuentra en un **88'9%** contra un **11'1%** que utilizaban iOS[1].

A pesar de que actualmente los *Smartphones* modernos son considerados como pequeños ordenadores, estos siguen teniendo una serie de limitaciones que deben tenerse en cuenta si uno quiere dedicarse a desarrollar aplicaciones móviles. La metodología para desarrollar una aplicación móvil es muy similar a desarrollar una aplicación de escritorio, pero deberán tenerse en cuenta las restricciones *hardware* que existen en estos dispositivos:

1 https://www.xatakamovil.com/sistemas-operativos/asi-como-android-se-ha-comido-mercado-diez-anos

- Pantallas más pequeñas y de menor resolución, aunque ya se dispone de resoluciones muy elevadas.
- Menor potencia de los procesadores.
- Memoria RAM limitada.
- Según el dispositivo, el almacenamiento también puede ser un inconveniente.
- Tasas de transferencia menores.
- Inestabilidad de las conexiones.

También es importante tener en cuenta las siguientes recomendaciones a la hora de comenzar el desarrollo de una **App**:

- Debido a la cantidad de marcas y modelos diferentes que se pueden encontrar en el mercado, en el campo de Android, debe buscarse la máxima compatibilidad entre todos ellos.
- Es importante crear un buen diseño de la aplicación para hacer un uso adecuado de los recursos, básicamente, liberarlos cuando estos no sean necesarios y no consumir más recursos de los debidos. Es recomendable hacer que la aplicación funcione bien en un modelo de gama baja, ya que esto implica que en un modelo de gama media/alta funcionará mejor.

2.2. Entornos de desarrollo integrados

Existen diferentes formas de poder desarrollar aplicaciones móviles, pero básicamente se distingue entre dos tipos de aplicaciones, **híbridas** o **nativas**. Las primeras son aplicaciones generadas mediante lenguajes *HTML5*, *JavaScript* y *CSS*, que utilizan un *plugin* tipo *Cordova*, independiente de la plataforma, que permite la comunicación entre la aplicación y la plataforma. También existen *frameworks* como **IONIC** o **Flutter** para *Dart*, que permiten generar este tipo de aplicaciones.

Las aplicaciones nativas son aquellas que se desarrollan en el lenguaje utilizado por la plataforma, por ejemplo ***Objective C*** o ***Swift*** para iOS y ***Java*** o ***Kotlin*** para Android. Las aplicaciones nativas por lo general ofrecen un mayor rendimiento con respecto a las aplicaciones híbridas, así como un mayor aprovechamiento de los recursos del propio dispositivo.

Durante el desarrollo de este libro se utilizará **Android Studio**[2] para crear aplicaciones móviles nativas, ya que se trata del IDE oficial para la plataforma Android, siendo ***Kotlin*** el lenguaje elegido.

2.2.1. Instalación y configuración de Android Studio

La documentación de Google para el desarrollo de aplicaciones Android es muy completa y de fácil consulta. Se comenzará por centrar la atención en la instalación de **Android Studio**[3], desde este enlace, https://developer.android.com/studio/index.html, desde donde podrás descargar la última versión.

2 Android Studio (https://developer.android.com/)
3 Documentación oficial de instalación de Android Studio (https://developer.android.com/studio/install)

> *En el momento de la revisión el texto se utiliza la última versión de **Android Studio** para **MacOS**, aunque muchos de los conceptos introducidos no varían con respecto a versiones anteriores. Debes tener en cuenta que este IDE está sometido a una evolución constante.*
>
> *No es necesario que utilices el mismo sistema operativo, puedes utilizar el sistema operativo que consideres más oportuno. Si vas a realizar la instalación sobre Linux te recomiendo que revises la documentación de instalación.*

Las aplicaciones nativas para **Android** se desarrollan utilizando como lenguaje de programación **Java**, o **Kotlin**. Por tanto, será necesario tener instalado en el SO el *software* necesario para la ejecución de **Java**. Se hace referencia a la máquina virtual de Java (JVM) y su entorno de ejecución de Java (JRE). Como mínimo, debes tener instalada la versión 8 de Java[4], aunque ya se recomienda el uso de la versión 11[5]. Puedes descargarlo desde la página de recursos técnicos de Oracle (https://www.oracle.com/technical-resources/).

Una vez instalado Java en el ordenador, el siguiente paso será descargar **Android Studio** para el sistema operativo con el que se vaya a trabajar.

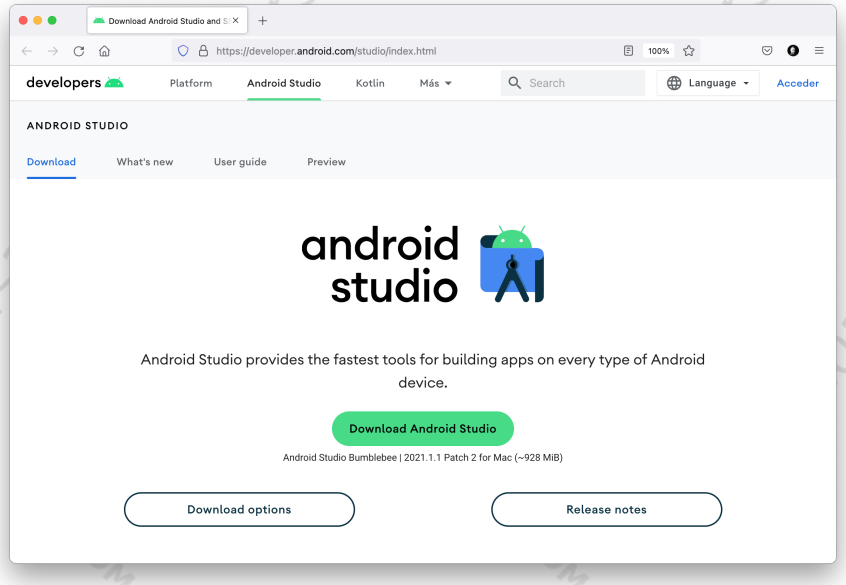
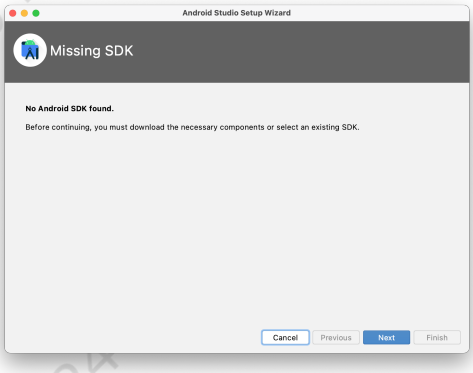
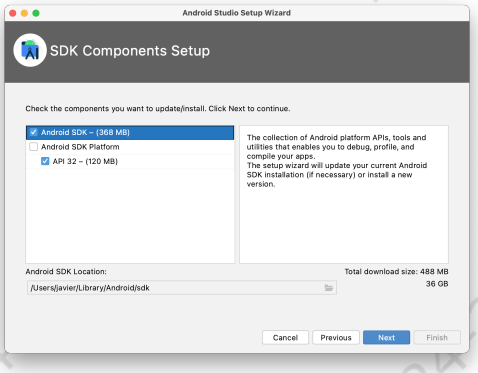


Figura 1

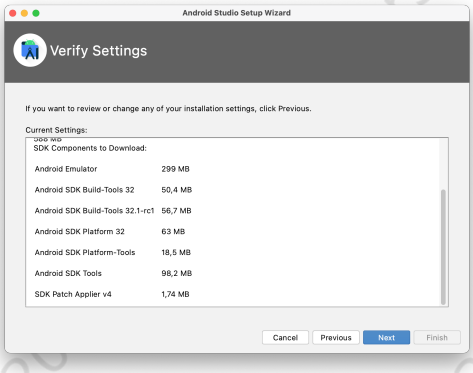
Una vez hayas descargado el instalador, ya puedes dar comienzo a la instalación del IDE en el sistema operativo. Como podrás ver a continuación, se trata de una instalación clásica guiada a través del asistente de instalación. Se han omitido algunos pasos del asistente, como el tipo de instalación, que suele seleccionarse *standard*, o la elección del tema, oscuro o claro.

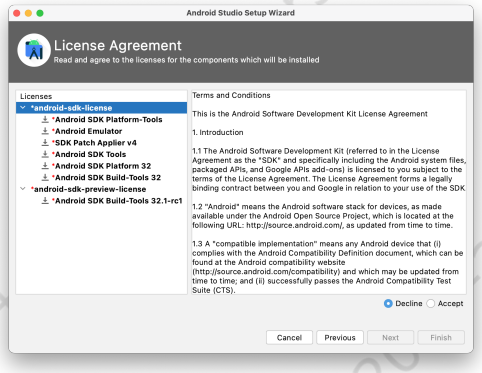
4 Java 8 (https://www.oracle.com/java/technologies/downloads/#java8)
5 Java 11 (https://www.oracle.com/java/technologies/downloads/#java11)



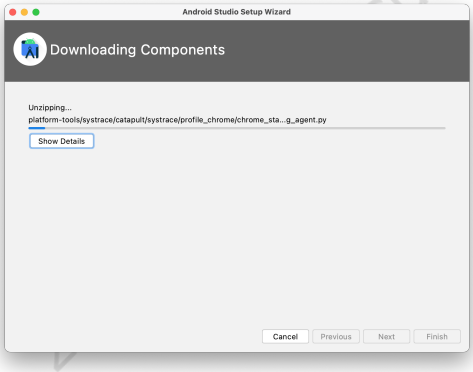


Se dejarán las opciones por defecto, si es una instalación limpia verás que detecta que no dispones de ningún SDK instalado, por lo que se descargará el último disponible.





Tras aceptar las condiciones se iniciará el proceso de instalación del entorno.



Una vez finalizada la instalación, ya puedes ejecutar el IDE. En primer lugar podrás ver en la parte izquierda las opciones ***Projects***, para crear y gestionar los proyectos que se vayan generando. La opción ***Customize***, esta permite acceder de manera rápida a opciones visuales y de accesibilidad, pero también encontrarás el acceso a todas las opciones de configuración del IDE.

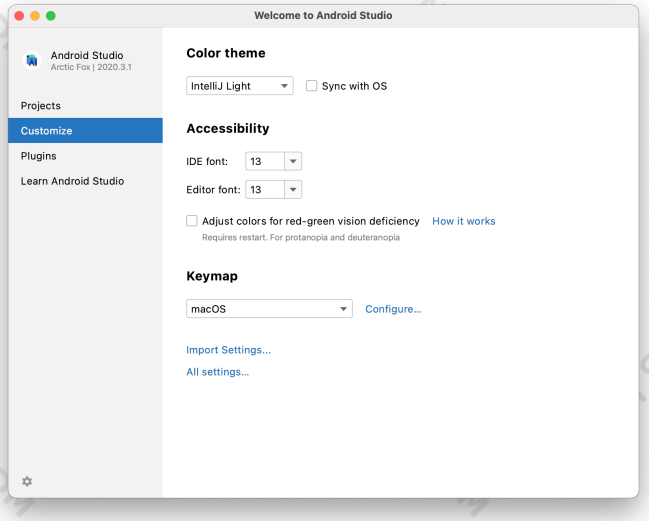


Figura 2

La opción ***Plugins***, desde la cual pueden añadirse y eliminarse *plugins* al entorno de desarrollo.

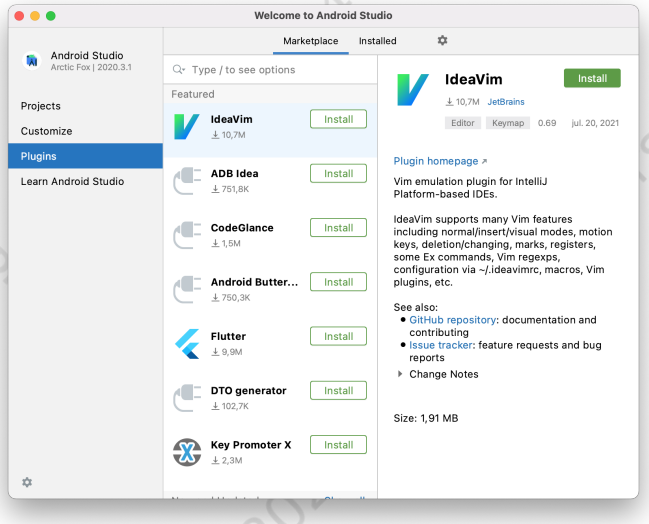


Figura 3

Y por último ***Learn Android Studio***, donde encontrarás enlaces a documentación, ejemplos y consejos sobre el IDE.

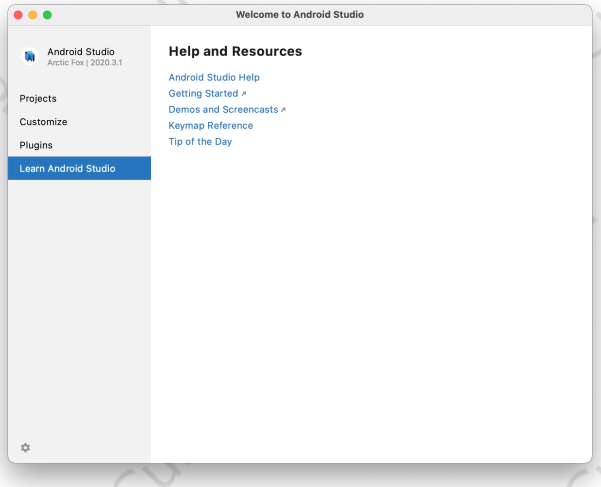


Figura 4

2.2.2. Android SDK

Ahora, se mostrará como añadir **SDK**s (*Software Development Kit*) de Android según se necesite. Tras la instalación, vendrá instalada la última versión disponible, pero si investigas un poco, verás que la última versión, no es siempre la más utilizada en el caso de Android. Desde ***Projects***, selecciona la opción ***SDK Manager***. Si la pantalla no aparece como la imagen siguiente, si no tienes proyectos creados, la encontrarás justo en el centro, en la opción ***More Actions***.

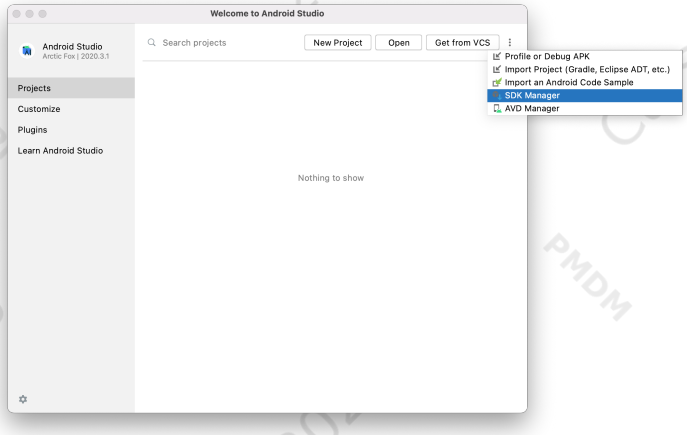


Figura 5

De momento no es necesario añadir ningún API al IDE, se utilizará la última que contiene todo lo necesario para poder trabajar.

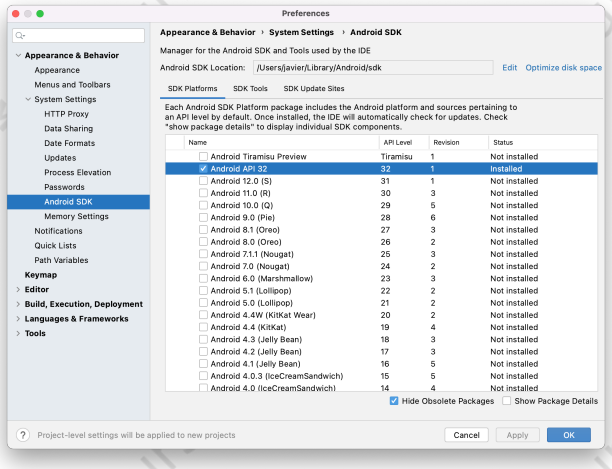
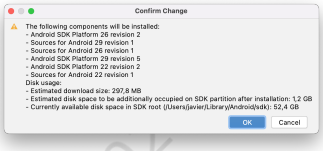
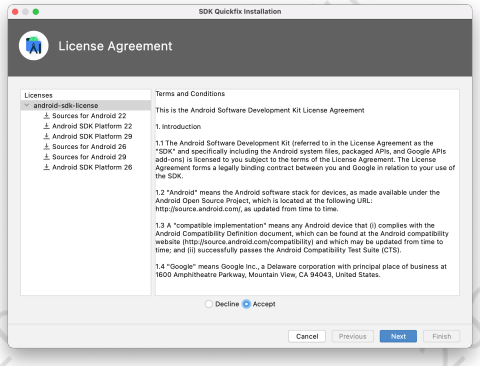


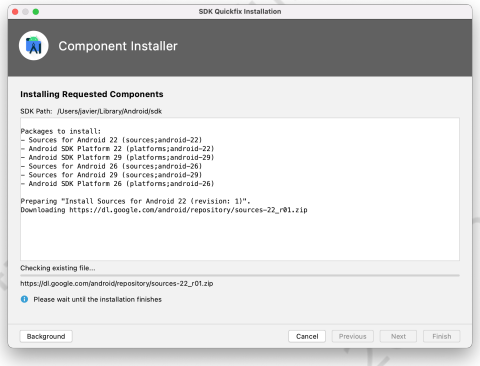
Figura 6

Si se necesitase añadir una API anterior, bastará con seleccionar la API deseada y pulsar el botón *Apply*, tras lo que se mostrará un mensaje informando de la descarga e instalación que va a producirse, deberás pulsar el botón *OK* para comenzar.



Seguidamente se deberá aceptar el contrato de licencia y pulsar el botón *Next* para comenzar con el proceso. A partir de aquí, **Android Studio** comenzará su trabajo, descargará e instalará los componentes seleccionados.





Una vez terminada la instalación, pulsa el botón *Finish*, seguidamente verás que aparecen marcadas las versiones instaladas. Ya podrías continuar preparando el entorno de trabajo.

2.2.3. Dispositivos de pruebas

Antes de comenzar a desarrollar, se terminará de configurar el entorno de desarrollo. Las pruebas, como bien sabrás, son una parte muy importante durante el desarrollo, es necesario saber si lo que se está haciendo funciona o no.

Se centrará la atención en el emulador **AVD** que incorpora **Android Studio** y en los dispositivos físicos. Existen otras alternativas en caso de no poder utilizar ninguno de estos dos medios, como las siguientes:

- BlueStacks 4: https://www.bluestacks.com/es/index.html
- Genymotion: https://www.genymotion.com/

Como último recurso para realizar pruebas, se puede generar un archivo de instalación de Android, fichero del tipo **.apk**, y enviarlo al dispositivo, esto se mostrará más adelante.

Emulador AVD

Como ya se ha comentado, **AVD** (Android Virtual Device) es emulador que viene por defecto con Android Studio, este permitirá simular características físicas de dispositivos reales, ya bien sean teléfonos, *tablets*, incluso dispositivos *Wear OS*, Android TV, etc.

La recomendación de Google es crear un AVD por cada una de las imágenes disponibles del sistema, pero en este caso, con una nos bastará, quiero hacer hincapié en el espacio en disco, ya que si además de tener instaladas varias APIs, debe crearse una imagen AVD por cada una …

Para acceder al administrador AVD, se seguirán los mismos pasos que los utilizados para acceder al *SDK Manager*, pero esta vez, selecciona la opción ***Virtual Device Manager***.

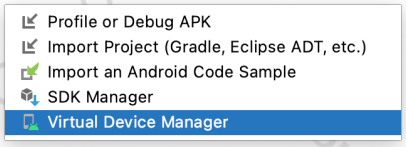


Figura 7

Una vez dentro, se mostraría el listado de los AVDs creados, en este caso al tratarse del primero, ese listado estará vacío y la única opción disponibles será ***Create Virtual Device…*** En versiones anteriores, en la parte inferior podías encontrar un enlace a *Android Dashboards*[6], donde podías encontrar estadísticas sobre los dispositivos utilizados, tamaños de pantalla, dispositivos que utilizan OpenGL, etc. Pero como ocurre con las APIs, que verás a continuación, estos datos no se actualizan desde agosto-septiembre de 2020.

6 *Android Dashboards* (https://developer.android.com/about/dashboards/index.html)

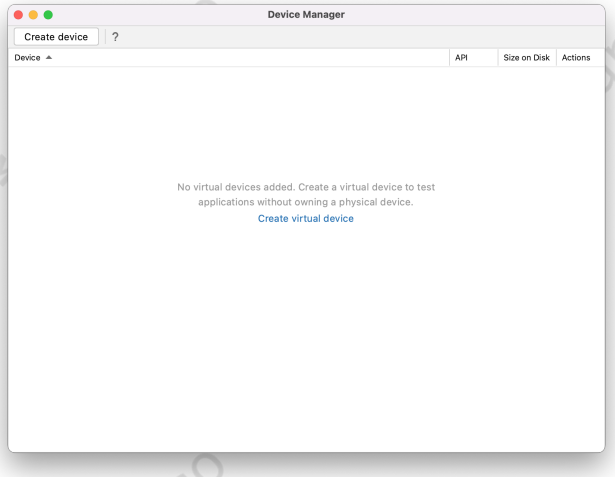


Figura 8

Una vez pulsada la opción ***Create Virtual Device***, o el botón ***Create Device***, se pasará a la selección del dispositivo, *Category*, en este caso se seleccionará *Phone*, y se seleccionará un modelo de la lista del centro, encontrarás modelos ofrecidos por Google (los *Pixel*), o modelos genéricos que encontrarás en la parte baja de la lista.

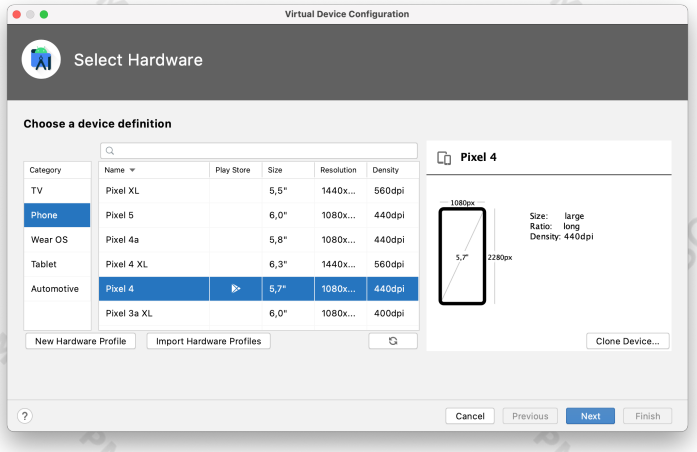


Figura 9

Se utilizará un ***Pixel 4*** con Servicios de Google Play[7], estos, al igual que en los dispositivos reales, permiten hacer uso de funciones avanzadas que como desarrollador, facilitan el uso de ciertos servicios, como el localización, *Places*, *Firebase*, etc. Su uso también hará que se necesiten permisos de administrador para realizar ciertas modificaciones sobre el dispositivo emulado.

7 *Google Play Services* (https://developers.google.com/android/guides/overview)

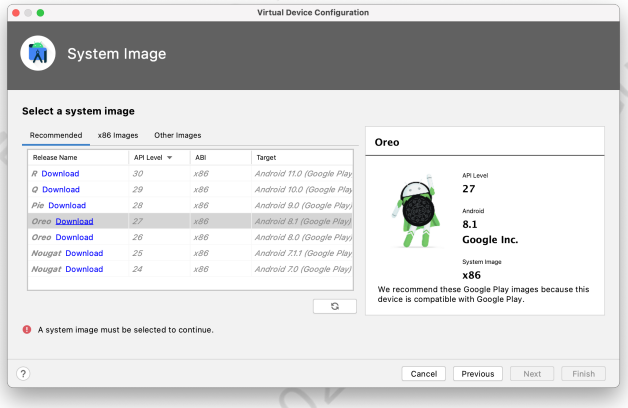


Figura 10

El siguiente paso será seleccionar la imagen del sistema, al tratarse de la primera, verás que todas requieren descarga. Se utilizará la ***Oreo 8.1 (API 27)***, no es la más utilizada, pero tampoco es la más moderna, de esta forma se trabajará en un término medio. Deberás descargarla para continuar.

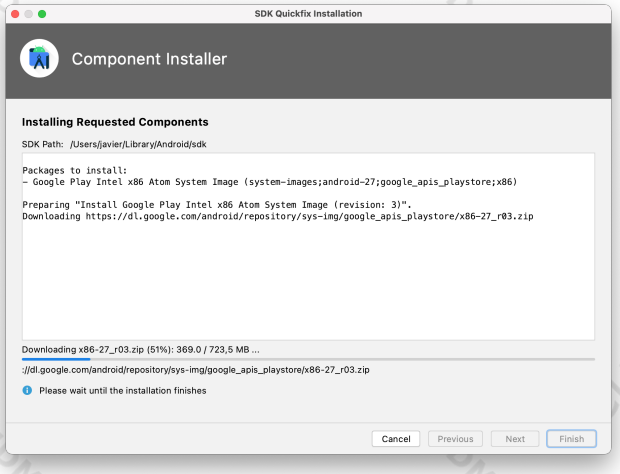


Figura 11

El siguiente paso será indicar el nombre del dispositivo, recomiendo indicar información de la API que tiene instalado para facilitar la elección, no será el caso ahora, pero se suelen tener más de dos o tres AVDs creados para las pruebas.

Una opción que puede venir bien es ***Enable Device Frame***, por defecto viene activada, pero se recomienda desactivarla para aligerar levemente el uso del emulador, sobretodo si tu equipo no tiene suficiente memoria.

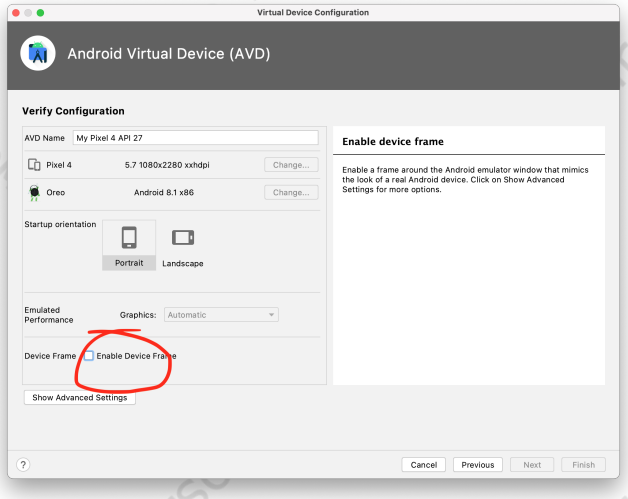
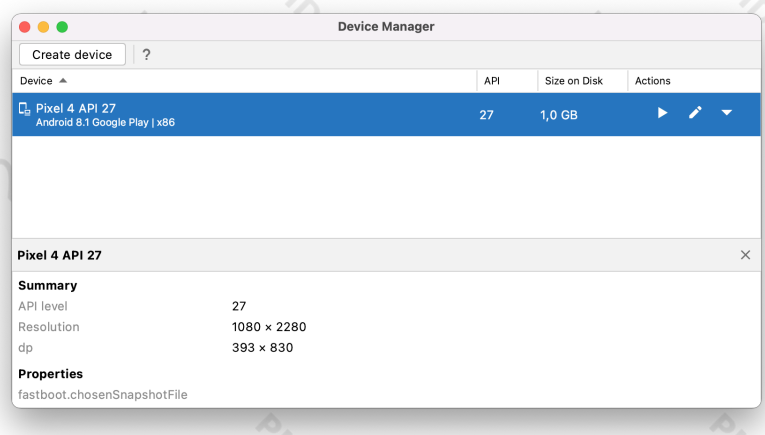
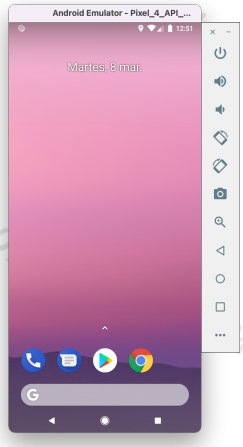


Figura 12

Una vez finalizado el proceso, el dispositivo aparecerá en el listado del *Device Manager*, ya solo faltará lanzarlo, utilizando el botón ***play*** que aparece en la columna *Actions*, para ver que funciona correctamente. Tras unos segundos de tensa espera, deberá cargarse el dispositivo en el emulador.





Una vez cargado, es posible que comience a actualizar aplicaciones como si de un dispositivo físico se tratase, puedes ajustarlo a tu manera, por ejemplo, poner el idioma Español para el sistema, etc.

Dispositivos reales

Los dispositivos reales ya son otra historia, por un lado, ayuda ver tu aplicación funcionando en un dispositivo real, parece más profesional, además, te permite liberar memoria en el equipo al no tener que cargar un AVD.

Si utilizas **macOS**, no tienes que hacer nada, al menos nunca se me ha planteado lo contrario, pinchas el dispositivo, AS lo reconoce y a funcionar. Eso sí, necesitarás activar el modo desarrollador en tu dispositivo.

Si no es tu caso, y trabajas con **Windows**, debes instalar el *driver USB*, en primer lugar prueba con el ***Google USB Driver*** que puedes instalar desde el gestor ***Android SDK***, el mismo que se utilizó para añadir las APIs, seguramente tendrás que reiniciar el sistema.

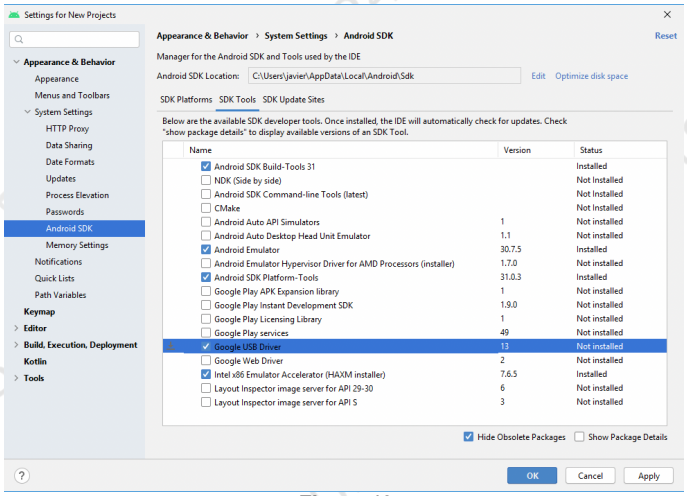


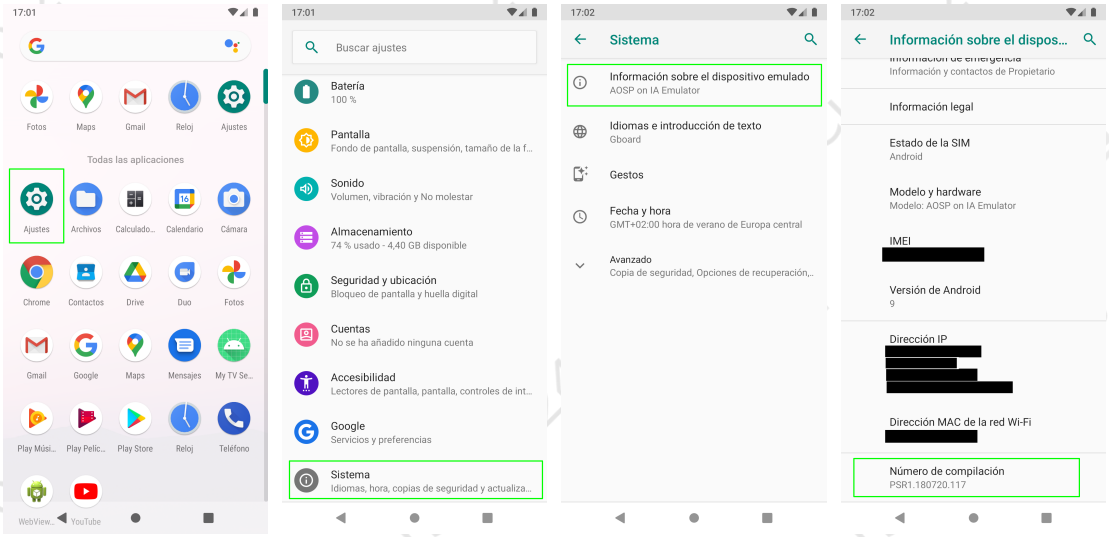
Figura 13

El *driver* de Google ha mejorado bastante y, salvo que tu dispositivo sea muy muy actual, no deberías tener problemas, pero… si fuese el caso, deberás acudir a la página del fabricante de tu dispositivo móvil y obtener el ***ADB driver*** para tu sistema operativo, por ejemplo, para un móvil Samsung → https://developer.samsung.com/mobile/android-usb-driver.html.

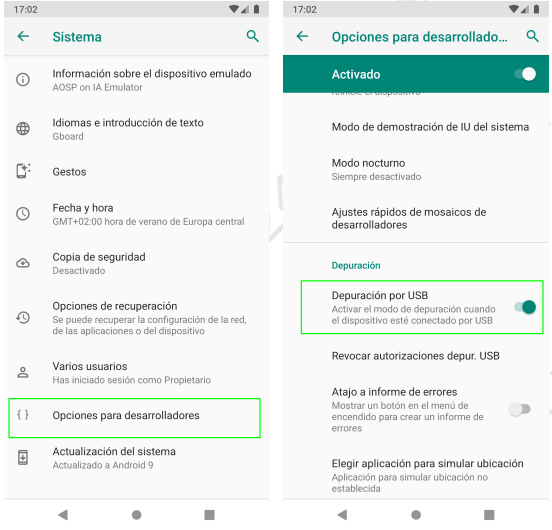
Activar el modo desarrollador

Si decides hacer uso de un dispositivo móvil, deberás activar el modo desarrollador para realizar pruebas y acceder a otras funciones, como la depuración, capturar pantalla e incluso grabarla.

Para acceder a las opciones para desarrolladores en un dispositivo Android, en primer lugar dirígete a los ajustes del dispositivo, busca la información del dispositivo y localiza el número de compilación. Una vez localizado el número de compilación, deberás pulsar sobre él siete veces, verás que aparece un mensaje indicando que las opciones para desarrollador están activadas.



Ahora, en la sección ***Sistema***, en las opciones avanzadas deberás buscar ***Opciones para desarrolladores*** y activar ***Depuración por USB***.



El dispositivo ya estaría listo para realizar las pruebas. Este proceso puede variar según el dispositivo, por ejemplo, en un dispositivo *Redmi 9*, deberás hacer las siete pulsaciones en la opción *"Versión de MIUI"*[8] para activar las opciones para desarrolladores.

8 MIUI (https://es.wikipedia.org/wiki/MIUI)

2.2.4. Creación del primer proyecto en Android Studio

Llegados a este punto, ya dispones del entorno listo para comenzar a desarrollar tus propias aplicaciones móviles. Para crear el primer proyecto, ***"Hello World!!"***, utilizando el botón ***New Project*** que tienes disponible desde la sección *Projects*.

Tras pulsar la opción para crear un nuevo proyecto se lanzará un asistente que te guiará durante la creación. Verás que no se trata de muchos pasos y es de fácil creación. El primer paso consistirá en seleccionar el tipo de aplicación, se creará una aplicación para móviles y *tablets* (***Phone and Tablet***) y se partirá de una actividad vacía (*Empty Views Activity*) como actividad principal, lo que vendría a ser una pantalla en blanco, más adelante se definirá que es una *Activity* en detalle.

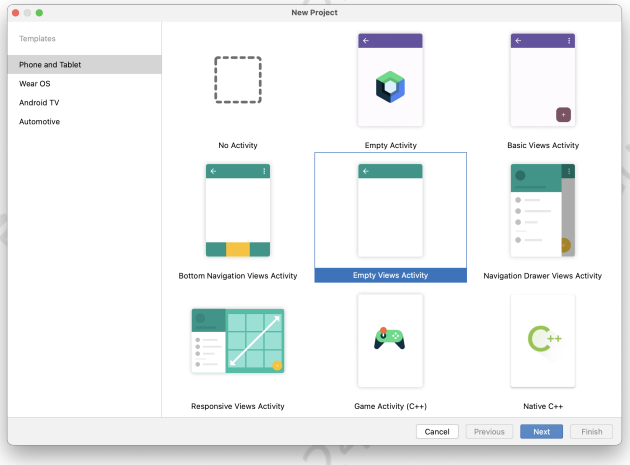


Figura 14

El siguiente paso consistirá en la configuración general del proyecto, deberás indicar el nombre (***Name***), el ***Package name***, este debe ser un nombre único que identifique la aplicación, su formato será **dominio.compañía.aplicación**. La localización del proyecto (***Save location***), por defecto se creará en la carpeta *AndroidStudioProjects* (puedes elegir el destino que prefieras). El lenguaje (***Language***) que se utilizará como base, **Kotlin** y, por último, la API mínima (***Minimum SDK***) que se utilizará, por lo general se utilizarán las que mayor porcentaje de uso tengan, pero se debe evaluar el uso de componentes. Se optará por un equilibro y se elegirá la API 24. Puedes utilizar la opción *Help me choose* para ver los porcentajes de uso de las APIs, pero Google lleva desde 2019 aproximadamente sin actualizar estas estadísticas.

También debes tener en cuenta a la hora de seleccionar el API, que cuanto más nueva sea, menos dispositivos podrán utilizar tu aplicación, por tanto, debes pensar si realmente utilizas funciones muy novedosas como para limitarte a APIs nuevas, o por el contrario, si es más de ámbito general sin muchas funciones, o ningunas, nuevas, utilizar una de mayor alcance.

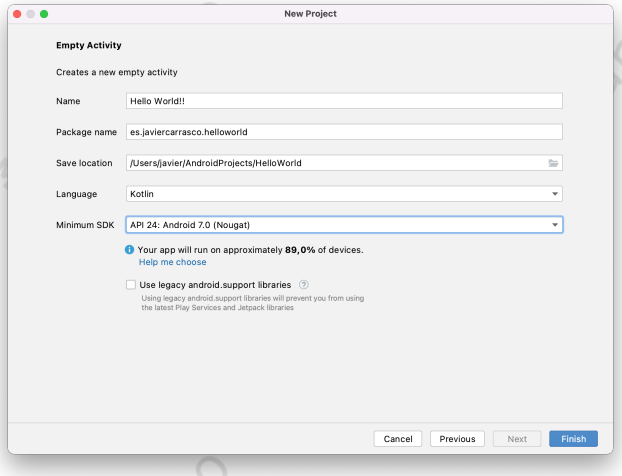


Figura 15

Con respecto a la opción *"Use legacy android.support libraries"*, esto hace que el proyecto utilice las librerías `android.support` [9] en vez de las librerías `androidx` para añadir compatibilidad con APIs anteriores, se evitará en la medida de lo posible su uso, ya que están *deprecated*.

Una vez terminado de configurar este paso, ya podrás pulsar el botón *Finish*. Tras unos segundos de carga, tendrás disponible el IDE con el proyecto listo para trabajar.

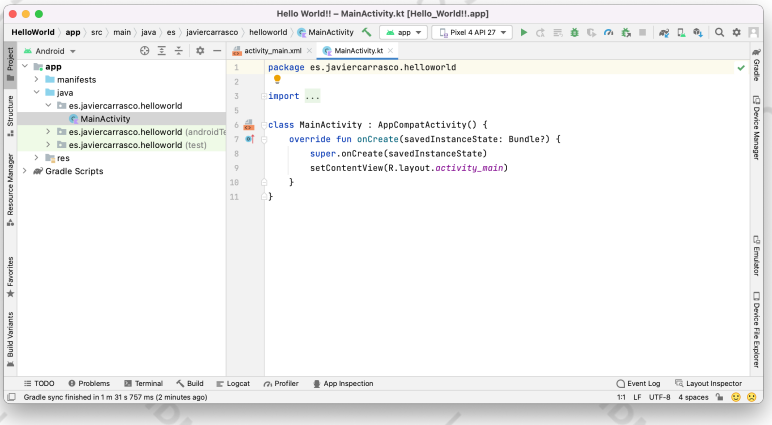


Figura 16

Tras la creación del proyecto, esta será la imagen del IDE que deberías ver. Ten en cuenta que la primera vez que se crea un proyecto puede tardar algo más en terminar de cargar, ya que necesita descargar librerías y sincronizar.

9 Bibliotecas de compatibilidad (https://developer.android.com/topic/libraries/support-library)

Antes de seguir, se harán una serie de observaciones sobre el funcionamiento básico de los proyectos Android y una primera toma de contacto con Kotlin.

Los proyectos creados en AS trabajan inicialmente utilizando el patrón MVC, si te fijas en la siguiente imagen, la clase principal del proyecto, ***MainActivity.kt***, hace la inyección de la vista utilizando el método ***setContentView()*** en la función ***onCreate()*** de la clase.

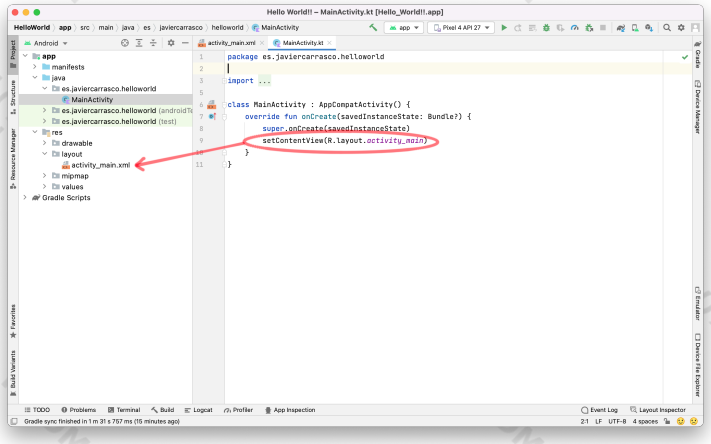


Figura 17

La vista, ***activity_main.xml***, es un archivo XML que contendrá la disposición de los elementos en pantalla, representa la interfaz de usuario. Esta vista puedes intercambiarla entre diseño, código y/o ambas utilizando los selectores que encontrarás en la esquina superior derecha.

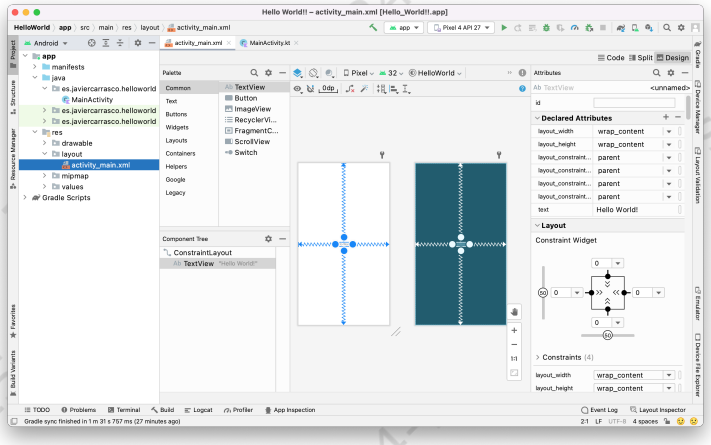


Figura 18

Esta vista es bastante intuitiva, la parte central muestra el estado actual de la vista, sobre la que puedes añadir los componentes que encontrarás en la paleta (parte izquierda).

Debajo de la paleta encontrarás el árbol de componentes añadidos a la vista. Seleccionando el componente, ya bien sea en el árbol, o en la vista, en la parte derecha verás todas las propiedades que pueden modificarse de cada elemento.

Observa que la clase principal, en código **Kotlin**, se encuentra dentro de la carpeta ***java***, nombre que se mantiene por romanticismo. Esta carpeta contendrá todas las clases que se vayan necesitando. La vista, por contra, se encuentra dentro de la carpeta ***res*** (*resources*), que se irá detallando a medida que se vaya avanzando. La primera que se utilizará es ***layout***, donde se crearán todas las vistas que tenga la aplicación. Fíjate ahora en algunas diferencias entre **Kotlin** y **Java** antes de comenzar.

```kotlin
package es.javiercarrasco.hellocefire

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

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

```java
package es.javiercarrasco.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Figura 19

Lo primero que observarás es la expresividad que ofrece Kotlin, reduciendo el número de palabras repetitivas, a medida que se avance, verás como se va reduciendo el código que se necesita escribir si lo comparas con Java.

Repasando este ejemplo, verás que en Kotlin por defecto todo es público, para heredar se omite la palabra *extends* y se sustituye por **":"**. Las funciones se declaran utilizando la palabra reservada ***fun*** y si no devuelven valor, no se indica (no se utiliza *void*). Y los puntos y coma ";" de Java, desaparecen. Todas y estas otras cosas que se irán viendo, le han hecho ganarse el mal llamado *Swift* para Android.

2.2.5. Hello World!!

Como ya se comentado, el lenguaje de programación que se utilizará será **Kotlin**. Este es un lenguaje de programación *tipado* que trabaja sobre la máquina virtual de Java (JVM), creado por *JetBrains* en 2011. Interopera con Java y dependiente del código Java de sus bibliotecas de clases, además, como se verá a medida que se vaya avanzando, tiene muchas semejanzas con Java.

En 2017, en la *Google I/O'17*, se anunció Kotlin como nuevo lenguaje oficial para el desarrollo de *apps* para Android, quedando totalmente integrado en Android Studio a partir de su versión 3.

Conociendo esto, se continuará con el proyecto *"Hello World!!"* creado en el punto anterior para la primera toma de contacto.

Ya se dispone en el proyecto de la primera *Activity*, la cual estará formada por una clase Kotlin, llamada `MainActivity.kt`, y un fichero XML llamado `activity_main.xml`. Se entenderán las ***Activities*** como las pantallas que se mostrarán a los usuarios de la aplicación en sus dispositivos, y como se puede ver, están divididas en dos partes, parte lógica y parte visual. La parte lógica es la clase Kotlin que contendrá el código necesario para añadir funcionalidad. La parte visual, conocida como *layout*, es el fichero XML asociado en el que se añadirán los componentes (botones, etiquetas, etc) que se quieran mostrar. Ambas partes se unen mediante el método ***setContentView()*** comentado anteriormente.

Se comenzará por modificar la vista, en el fichero `activity_main.xml`, realiza los siguientes cambios. En el ***TextView*** (es una etiqueta), modifica la propiedad ***Text***, escribe el texto *"Hello World!!"*. En la propiedad ***textAppearance*** selecciona *TextAppearance.AppCompat.Display1*. Y modifica su ***Vertical Bias*** a 10 como se muestra en la figura.

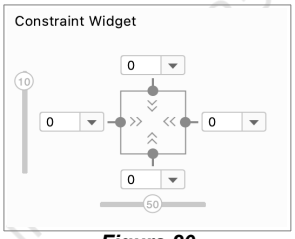


Figura 20

Te habrás fijado que una vez modificado el texto de la etiqueta, aparece un *warning* de ***hardcoded text***. Esto es debido a que en Android, para facilitar la implementación de diferentes idiomas, y evitar el texto fijo en código, se hace uso del fichero `strings.xml` que encontrarás en ***res/values***. Lo que deberás hacer es editar el fichero y añadir la variable ***hola_Mundo*** con el texto que has puesto en la propiedad *Text* de la etiqueta, pero esta vez, en castellano. Verás que ya existe una variable con el nombre del proyecto.

```
<resources>
    <string name="app_name">Hello World!!</string>
    <string name="hola_Mundo">¡Hola Mundo!</string>
</resources>
```

Ahora ya puedes cambiar la propiedad *Text* y poner la variable creada en *strings.xml (@string/hola_Mundo)*, de esta forma se evita este tipo de *warnings*.

Añade un segundo *TextView* justo debajo del primero, llámalo ***tv_saludo***, elimina el texto de la propiedad *Text* y añade el texto *"Saludo"* a la propiedad ***tools:text*** (la que tiene una llave inglesa a su izquierda). Relaciona el *constraint* superior con la primera etiqueta y el inferior con el *parent*, modifica su *Vertical Bias* a 10 cuando ajustes sus *constraints* como se muestra en la imagen. Si te fijas, los *constraints* laterales de la nueva etiqueta se unen a los laterales de la etiqueta superior, de esta manera, con solo centrar un elemento, el resto se ajustará a sus cambios.

Figura 21

Añade ahora un componente ***PlainText*** (para introducir texto), con *id* ***et_saludo***, ajusta el alineado igual que la segunda etiqueta, pero esta vez coge como referencia la segunda etiqueta y su *Vertical Bias* a 20. La propiedad ***inputType*** déjala como *textPersonName*, observa las opciones, según la opción (u opciones) que elijas, no se filtrará el texto introducido, pero el sistema operativo mostrará el teclado que más se adapte. Elimina la propiedad *Text* y crea una nueva variable en *strings.xml* que se llame ***hint_nombre*** con el texto *"Tu nombre aquí"*, añade esta variable a la propiedad ***hint***.

Por último, añade un ***Button***, indica como *id* ***btn_saludar*** y aplica el mismo alineado que el empleado con el *PlainText*, pero ahora utilizando como referencia el propio *PlainText*. Crea una nueva variable en *strings.xml* que se llame ***txt_boton*** con el texto *"Saludar"* y añádelo a la propiedad *Text* del botón.

El resultado que se pretende obtener mediante esta configuración puedes verlo en la imagen mostrada a continuación.

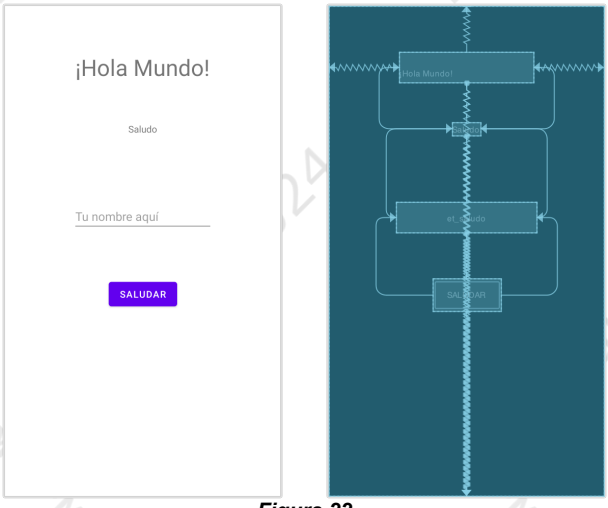


Figura 22

El código XML resultante puedes verlo a continuación, por ahora, si no te aclaras con el diseñador, puedes copiar y pegar para no perder mucho tiempo. En la esquina superior derecha del diseñador encontrarás las opciones para intercambiar entre diseño y código.

```xml
<?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"
    tools:context=".MainActivity">

     <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hola_Mundo"
        android:textAppearance="@style/TextAppearance.AppCompat.Display1"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

```
        app:layout_constraintVertical_bias="0.10" />

    <TextView
        android:id="@+id/tv_saludo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/textView"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.10"
        tools:text="Saludo" />

    <EditText
        android:id="@+id/et_saludo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="@string/hint_nombre"
        android:inputType="textPersonName"
        android:minHeight="48dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/tv_saludo"
        app:layout_constraintStart_toStartOf="@id/tv_saludo"
        app:layout_constraintTop_toBottomOf="@+id/tv_saludo"
        app:layout_constraintVertical_bias="0.20" />

    <Button
        android:id="@+id/btn_saludar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/txt_boton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/et_saludo"
        app:layout_constraintStart_toStartOf="@+id/et_saludo"
        app:layout_constraintTop_toBottomOf="@+id/et_saludo"
        app:layout_constraintVertical_bias="0.20" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Ahora sí, vamos a ver el código Kotlin, ve a la clase ***MainActivity.kt***. La idea es que al pulsar el botón saludar, en la etiqueta saludo aparece un saludo personalizado, siempre y cuando el cuadro de texto tenga un nombre.

Si antes de cambiar nada, lanzas la aplicación utilizando el botón ***Run 'app'*** de la barra de navegación (si tienes conectado un dispositivo real este aparecerá seleccionado por defecto, si despliegas, podrás ver los dispositivos virtuales, selecciona el dispositivo sobre el que ejecutar y pulsa el botón *Run*, en el desplegable de la izquierda deberá estar seleccionada la opción *app)*…

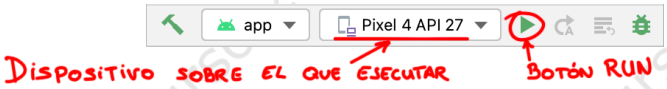


Figura 23

…deberías obtener algo similar a lo que se muestra en la siguiente imagen.

Figura 24

En la clase principal, lo primero que deberá hacerse es inyectar los elementos objeto de uso, la etiqueta saludo, el cuadro de texto y el botón. Crea las siguientes tres propiedades en la clase principal.

```
1  private lateinit var tvSaludo: TextView
2  private lateinit var etSaludo: EditText
3  private lateinit var btnSaludar: Button
```

Como puedes ver, las declaraciones en Kotlin son algo diferentes a las de Java. Por defecto, todo es público, por tanto, si quieres que algo sea privado debes declararlo.

La declaración de variables comienza por ***var*** o ***val***, según sea la declaración de una **var**iable o un *final*(“***val**ue*”), es decir, mutable o inmutable (se puede o no modificar) respectivamente. El tipo de dato se especificará después del nombre, separado por dos puntos “:”.

Kotlin tiene los tipos de datos **inferidos**, es decir, no es necesario especificar el tipo de variable si se asigna un valor en la declaración, no siendo ambiguo para el compilador.

Además, requiere la inicialización de las variables en el momento de la instanciación, para evitar esta obligatoriedad, con variables que se crean al inicio, pero que se inicializan más adelante, se utiliza la palabra reservada ***lateinit*** delante de ***var*** y/o ***val***.

Ahora, en el método ***onCreate()*** se instanciarán las variables creadas con los componentes de la vista.

```
tvSaludo = findViewById(R.id.tv_saludo)
etSaludo = findViewById(R.id.et_saludo)
btnSaludar = findViewById(R.id.btn_saludar)
```

En este caso, se hace referencia a cualquier identificador creado en los *layouts* mediante el método ***findViewById()***, más adelante se verá que existen mejores formas de hacer esta inyección.

El siguiente paso es crear un ***listener*** para el botón, este permitirá ejecutar acciones cuando el usuario pulse sobre él. La idea es comprobar que el cuadro de texto contenga texto y así poder saludar. Añade las siguientes líneas al método *onCreate().*

```
btnSaludar.setOnClickListener {  it: View!
    tvSaludo.text = "Hola " + etSaludo.text
}
```

Evidentemente, con este código, no se comprueba si la propiedad *text* del cuadro de texto contiene información y, además, se está creando un *hardcoded text* al escribir el texto directamente en un literal. Observa una cosa muy interesante, y es que Kotlin, es un gran aficionado al uso de *lambdas*. El objeto *it* que aparece sombreado es el componente que devuelve la llamada, en este caso, el propio botón, y puedes utilizar en cualquier momento dentro de la *lambda*, a continuación se hará una breve introducción al uso de este objeto.

Para solucionar el *hardcoded text* se creará la siguiente variable en *strings.xml*, de esta forma también podrás ver como acceder a los recursos desde código.

```
<string name="texto_saludo">¡¡Hola %s!!</string>
```

El ***%s*** se sustituye por cualquier *string*, si fuese ***%d***, se esperaría un entero. Modifica ahora el *listener* para acceder a los recursos *strings*.

```
btnSaludar.setOnClickListener {
    tvSaludo.text = resources.getString(R.string.texto_saludo, etSaludo.text)
}
```

Para acceder a los recursos debes utilizar el objeto ***resources*** y obtener el identificador del recurso, para ello se utiliza la clase ***R***, habrás visto que al escribir el punto tras la R aparece la lista de recursos a los que puedes acceder.

Ahora se comprobará que el campo que de debe contener el nombre no esté vacío, si no, en la etiqueta se mostraría *"¡¡Hola !!"*, y no quedaría bien, se creará una nueva variable llamada *texto_error* en el recurso *strings.xml*.

```
<string name="texto_error">El nombre no puede estar vacío</string>
```

Modifica el *listener* del botón añadiendo el siguiente código.

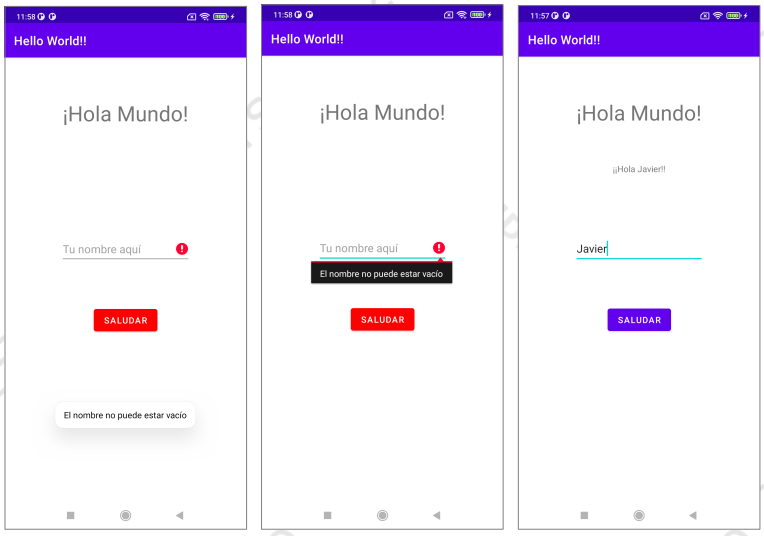
```
btnSaludar.setOnClickListener { btn ->
    if (etSaludo.text.isBlank()) {
        etSaludo.error = resources.getString(R.string.texto_error)
        Toast.makeText(this, R.string.texto_error, Toast.LENGTH_SHORT).show()
        btn.setBackgroundColor(Color.RED)
    } else {
        etSaludo.error = null
        tvSaludo.text = resources.getString(R.string.texto_saludo, etSaludo.text.trim())
        btn.setBackgroundColor(resources.getColor(R.color.purple_500))
    }
}
```

En primer lugar se ha cambiado el nombre al objeto ***it*** por ***btn***, al hacerse esto debe añadirse la flecha *lambda* (→). La estructura del ***if***, comprueba el si el contenido de la propiedad *Text* del cuadro de texto no está en blanco con ***isBlank()*** (se podría utilizar *isEmpty()*, pero si añades solo espacios no saltaría el error).

Los *EditText* en Android tienen una propiedad llamada ***error***, a la cual, si se le asigna un texto se mostrará un aviso cuando se pulse sobre la exclamación de aviso que aparecerá.

También se introduce un nuevo elemento, el ***Toast***, son mensajes flotantes que se mostrarán al usuario en pantalla durante un determinado periodo de tiempo. Desaparecerá automáticamente y el usuario no interviene para nada. También pueden ser una buena herramienta para depurar el código.

Por último, puedes ver como se cambia el color de fondo del botón mediante la clase *Color* al producirse un fallo y, cuando es correcto accediendo al recurso color del proyecto (*colors.xml*). Seguramente el método *getColor()* te aparecerá *deprecated*, esto sucedió a partir de la API 23.



Cuando te encuentras con un *deprecated*, es recomendable buscar una alternativa actualizada o hacer un control de versión para evitar posibles problemas. Con el control versión se comprueba sobre que API se está ejecutando, de tal forma que si en esa versión no está *deprecated* se ejecuta tal cual, en caso contrario, se ejecutaría con la nueva versión.

```
if (Build.VERSION.SDK_INT >= 23) {
    btn.setBackgroundColor(this.getColor(R.color.purple_500))
} else {
    btn.setBackgroundColor(resources.getColor(R.color.purple_500))
}
```

Como sabemos que está *deprecated* desde la API 23, se puede comprobar como aparece en la condición del *if*. Aunque también se pueden utilizar constantes y utilizar esta condición.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) { // API 23 → M
```

2.2.6. Generar un fichero APK

Imagino que ya lo sabrás, pero si no es el caso, los archivos de instalación tienen la extensión **APK**. Cuando se genera un APK, el proyecto no debe tener errores. Si no vas a subir tu aplicación al *Play Store*, puedes generar un APK sin firmar y pasarlo a un dispositivo real para instalarlo, debes tener habilitada la opción que permita la instalación de terceros (orígenes desconocidos).

Para generar el fichero APK deberás utilizar la siguiente opción del menú ***Build > Build Bundle(s) / APK(s) > Build APK(s)***. El fichero generado lo podrás localizar dentro de la carpeta del proyecto en la ruta ***PROYECTO\app\build\intermediates\apk\debug\app-debug.apk***.

Puedes renombrar el fichero y pasarlo por USB al dispositivo para instalarlo, o incluso mandártelo por correo al teléfono.

2.3. Estructura de un proyecto Android

Es necesario saber que un proyecto de **Android Studio** puede contener diferentes módulos en función del tipo de aplicación que se quiera desarrollar, o la misma aplicación con diferentes APIs mínimas, o un módulo para cada uno de los dispositivos destino. Pero de momento, se pondrá el foco en el desarrollo de aplicaciones móviles.

La formación de los proyectos Android mediante módulos permite alcanzar independencia entre las distintas partes de un proyecto. Cada uno de los módulos deberá contener un descriptor de aplicación o *manifest*, éste se encontrará en la carpeta ***manifest***, donde se ubicará el fichero ***AndroidManifest.xml***. Este fichero define la aplicación o módulo. Describe su nombre, paquete, iconos, estilos, etc. Además, también se indicarán las *activities*, los *intents*, servicios, etc. Así como la versión mínima de Android para ejecutarla, la versión de la aplicación, etc.

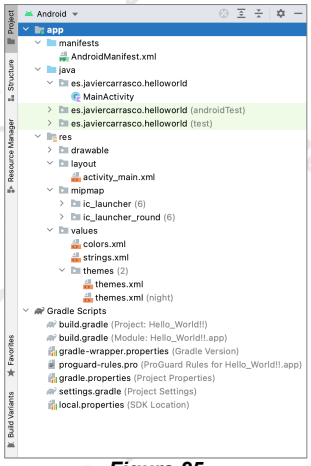


Figura 25

Para ampliar información sobre la estructura de un proyecto puedes consultar la documentación oficial[10].

La carpeta ***java*** contendrá todo el código fuente de la aplicación. El funcionamiento es similar al que puedas haber utilizado con *NetBeans* o *Eclipse*, almacenando los ficheros en paquetes. A pesar de su nombre, se encontrará tanto código en *Java* como código en *Kotlin*.

Dentro de la carpeta *java* se encuentra la clase ***MainActivity*** con el código, en este caso, *Kotlin*, fíjate en el icono con la ***K***. También habrá dos paquetes con las clases preparadas para insertar pruebas utilizando *JUnit*.

Seguidamente se encuentra la carpeta ***res***, esta contendrá todos los recursos utilizados por la aplicación. Si se examina su contenido se encontrarán las siguientes sub-carpetas:

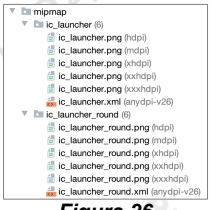


Figura 26

- ***drawable***, aquí se almacenarán los ficheros de imagen que utilizará la aplicación internamente, JPG, PNG o GIF, y los descriptores de las imágenes en XML[11]. En ocasiones también podrían almacenarse sonidos.
- ***layout***, aquí se encuentran todos los ficheros XML con las vistas de la aplicación. Estas vistas son las que definirán las interfaces de las pantallas[12].
- ***mipmap***, similar a la carpeta *drawable*. En *mipmap* únicamente se ubicarán los iconos que se utilicen para el lanzamiento de la aplicación, y en *drawable* el resto de recursos gráficos. En *mipmap* cada icono se almacenará en diferentes densidades, por ejemplo *hdpi*, para dispositivos con una densidad gráfica alta.
- ***values***, esta ubicación se utiliza para establecer los valores que se usen en la aplicación, de forma que, si se debe modificar algún dato, será aquí y, no se tendrá que estar navegando por el código fuente. En *colors.xml* se definen los tres colores principales de la aplicación. En *strings.xml* se definen todas las cadenas de texto de la aplicación, esto facilita la traducción de la aplicación. Si existe el fichero *dimens.xml*, ya que se puede crear a nuestra elección, se podrán guardar aquellas dimensiones que más se utilicen en la aplicación.
- ***themes***, dentro de *values*, se guarda, a modo de *pack*, los temas y estilos de la aplicación, por ejemplo *Theme.HelloWorld*, bajo el nombre de *themes.xml*. También podrás observar que existe un *themes.xml (night)*, que se utilizará al cambiar al modo oscuro.

Además de estas, en res pueden aparecer otras carpetas que podrán sernos de utilidad, como ***animator***, ***anim***, ***menu***, ***navigation***, ***arrays***, ***raw*** o ***xml***. Algunas de estas se verán más adelante.

Por último, la carpeta ***Gradle Scripts***. Esta carpeta contiene los ficheros con toda la información necesaria para compilar y construir la aplicación. La mayor parte de los ficheros hacen referencia al proyecto en general, otros referencian al módulo **app**. El fichero más importante será ***build.gradle (Module: app)***, donde se configurarán las opciones de compilación. Los parámetros que se pueden modificar, siempre con cuidado, son:

10 Descripción general del manifiesto de una app (https://developer.android.com/guide/topics/manifest/manifest-intro)
11 Drawable Resource (https://developer.android.com/guide/topics/resources/drawable-resource)
12 Layout Resource (https://developer.android.com/guide/topics/resources/layout-resource)

- ***compileSdk***: define la versión para la que se compila la aplicación.
- ***applicationId***: coincidirá con el nombre del paquete creado, se utilizará como identificador único de la aplicación.
- ***minSdk***: indica el nivel mínimo de API que la aplicación requiere.
- ***targetSdk***: indica la versión de API más alta con la que se ha probado la aplicación.
- ***versionCode*** y ***versionName***: indican la versión de la aplicación. Cada vez que se actualice la versión de la aplicación el valor de *versionCode* deberá incrementarse en uno, y *versionName* según la importancia de la actualización, por ejemplo, para una actualización menor podría ser *"1.1"* y para una versión mayor *"2.0"*.

Dentro de `buildTypes` se añaden otras configuraciones según el tipo de compilación que se quiera (*release* para distribución, *debug* para depuración, etc).

El último apartado, `dependencies`, es importante, aquí se añadirán todas aquellas librerías de compatibilidad adicional que se necesiten para el proyecto.

2.4. Ciclo de vida de una aplicación

Android está enfocado al funcionamiento en dispositivos móviles y, estos cuentan con una serie de características distintas a las que pueda tener un ordenador de sobremesa. Ya se han visto las limitaciones que esto supone, pero además, durante el uso de una aplicación móvil, puede ocurrir que se reciba una llamada, interrumpiendo el uso de la aplicación y, tras acabar la llamada, generalmente se vuelve al estado en el que se encontraba la aplicación que se estaba utilizando.

En Android se utiliza una interfaz de pantalla completa, en la que se permite el uso de notificaciones que tapan parcialmente la actividad que se tenga activa. También es posible compartir *Activities*, por ejemplo, si ya se tiene una actividad que muestra un mapa, se puede volver a hacer uso de ella sin necesidad de tener que cargarla de nuevo. Además, el usuario puede pulsar el botón *back* (volver atrás) en cualquier momento para ir a una *Activity* anterior. Esto se conocerá como *Activities Stack*, pila de actividades, la cual estará formada por un conjunto de pantallas apiladas por las que el usuario podrá moverse.

La acción de lanzar una nueva *Activity*, o recuperar una que no está visible, se conoce como ***Intent*** (intento o solicitud), por tanto, un *Intent* será el objeto responsable de lanzar una nueva actividad o pantalla. Las aplicaciones se pueden encontrar en diferentes estados:

- **Activa** o en ejecución, momento en el que ocupa el primer plano de la interfaz, siendo visible para el usuario y tiene el foco de la interacción.
- **Pausada** es cuando ha perdido el foco pero es parcialmente visible, generalmente cuando aparece un cuadro de diálogo sobre ella.
- **Detenida** será cuando ya no es visible en la interfaz.

El sistema podrá finalizar una *Activity* en cualquier momento, matando el proceso o utilizando el método `finish()`. También es importante saber que, cuando los recursos escaseen, el sistema finalizará las *Activities* empezando por aquellas que estén detenidas, después las pausadas y, en una situación crítica, las activas.

Cuando una *Activity* termina deberá guardar el estado en que se encuentra para que cuando se lance de nuevo se pueda recuperar. Cuando se produzca un cambio en el estado se invocarán a determinados métodos para que puedan realizarse las operaciones oportunas necesarias. Estos métodos son:

- ***void onCreate(Bundle savedInstanceState)***, llamada cuando se crea la *Activity*, es el lugar donde irá la inicialización de la aplicación.
- ***void onStart()***, se llama después de *onCreate()*, u *onRestart()* si ha estado parada y vuelve a estar visible para el usuario. Suele utilizarse para cargar elementos gráficos y/o animaciones.
- ***void onRestart()***, método llamado tras *onStop()*, cuando la *Activity* se prepara para ser mostrada al usuario. Generalmente cuando el usuario navega hacia atrás.
- ***void onResume()***, se llama después de *onRestart()* o *onPause()*, para que la *Activity* empiece a interactuar con el usuario. Es un indicador de que el *Activity* comienza a estar activa y lista para recibir entradas. Es el nivel más alto de la pila y visible al usuario.
- ***void onPause()***, cuando una actividad pasa un tiempo de inactividad pero sigue siendo visible en la pantalla. También se activa cuando otra *Activity* ocupa el primer plano.

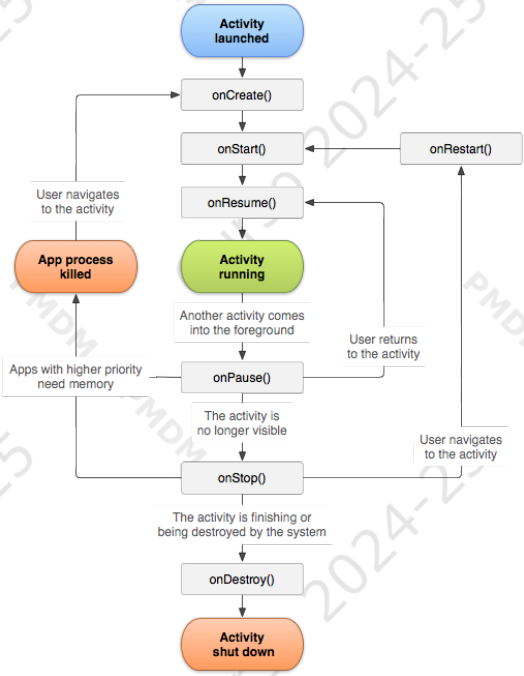


Figura 27

- ***void onStop()***, cuando pasa un tiempo que la *Activity* no es visible para el usuario, las siguientes llamadas serán a *onRestart()* o *onDestroy()*.
- ***void onDestroy()***, para liberar recursos antes de la destrucción de la *Activity*.

El parámetro ***savedInstanceState*** que aparece en el método *onCreate()* sirve para almacenar el estado de la instancia. Éste se utilizará para restaurar el estado previo de la *activity* cuando se vuelve a ella. La información almacenada en un objeto de tipo *Bundle* será un conjunto de pares tipo clave-valor. Puedes ampliar información acerca del ciclo de vida[13] de una *activity* en la documentación oficial de Google.

13 Ciclo de vida de una actividad (https://developer.android.com/guide/components/activities/activity-lifecycle)

De vuelta al proyecto *"Hello World!!"*, en la clase `MainActivity.kt`, en el método ***onCreate()*** añade la siguiente línea tras el método *setContentView()*.

```
Log.d("onCreate", "Activity creada!!")
```

Ahora, sobrecarga los siguientes métodos que representan los estados de una *activity*, puedes ir uno a uno, o utilizar la opción ***Override Methods*** del menú ***Code***, y seleccionarlos uno a uno de la sección *android.app.Activity*.

```
override fun onStart() {
    super.onStart()
    Log.d("onStart", "Método onStart()")
}

override fun onResume() {
    super.onResume()
    Log.d("onResume", "Método onResume()")
}

override fun onRestart() {
    super.onRestart()
    Log.d("onRestart", "Método onRestart()")
}

override fun onPause() {
    super.onPause()
    Log.d("onPause", "Método onPause()")
}

override fun onStop() {
    super.onStop()
    Log.d("onStop", "Método onStop()")
}

override fun onDestroy() {
    super.onDestroy()
    Log.d("onDestroy", "Método onDestroy()")
}
```

Si ejecutas la aplicación con los nuevos métodos sobrecargados, y activas la pestaña inferior ***Logcat***, podrás ver los mensajes creados utilizando la clase ***Log***, de tal forma que podrás hacer un seguimiento del ciclo de vida de las *activities*. Este es otro buen método para *debuggear* (*Log.d*) durante el desarrollo, así como mostrar *warnings* (*Log.w*) o mensajes de error (*Log.e*).

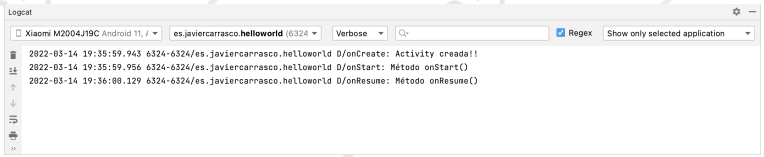


Figura 28

2.5. Conceptos importantes

Para terminar, se verán y repasarán algunos conceptos que se deben tener claros a la hora de desarrollar aplicaciones móviles para Android.

- ***Activity***, como ya se ha mencionado, una *Activity* es una pantalla de usuario, una interfaz, también lo verás mencionado como UI (*User Interface*). Está dividida en dos, parte lógica (una clase Java o Kotlin) y parte visual (*layout*).

- ***Fragment***, este es un concepto que se tratará más adelante, vienen a ser como una parte de la UI o comportamiento dentro de la misma. Puede pensarse en los *fragments* como en una sección parcial de una *activity*, con su propio ciclo de vida, que se incluye en otra *activity*.

- ***Services***, son componentes que se ejecutarán en segundo plano, realizando operaciones prolongadas o tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario. Un ejemplo podría ser, un servicio que reproduce música en segundo plano mientras se utiliza otra aplicación.

- ***Intents***, como ya se ha comentado, son las intenciones de lanzar una *Activity*, pero si se profundiza más en la definición, se puede decir que son mensajes asíncronos mediante los cuales se pueden lanzar *Activities*, *Services* y *Broadcast Receivers*[14].

- ***Content Providers***, son uno de los principales bloques de construcción de aplicaciones Android, proporcionan contenido a las aplicaciones. Encapsulan los datos y los proporcionan a la aplicación mediante la interfaz *ContentResolver*. Solo es necesario un *Content Provider* si se necesita compartir información con otras aplicaciones. Por ejemplo, los datos de los contactos son utilizados por una gran cantidad de aplicaciones, por lo que deben almacenarse en un proveedor de contenido. Si no se necesita compartir datos con otras aplicaciones puede utilizarse una base de datos a través de *SQLiteDatabase*.

- ***Broadcast Receivers***[15], son aquellos componentes encargados de recibir y responder a eventos globales generados por el sistema, un aviso de batería baja, una llamada, etc y también a eventos que produzcan otras aplicaciones.

14 Aspectos fundamentales de la aplicación (https://developer.android.com/guide/components/fundamentals)
15 *Broadcast Receivers* (https://developer.android.com/guide/components/broadcasts.html#receiving-broadcasts)

2.5.1. Contexto de una aplicación

El contexto de una aplicación, ***Context***[16], es una interfaz que permite el acceso a la información global del entorno de la aplicación. Es una clase abstracta proporcionada por el sistema operativo Android. Mediante esta, se permite el acceso a las clases y recursos de la aplicación, así como permitir llamadas ascendentes a nivel de aplicación, como lanzar actividades, difusión y recepción de *Intents*.

Simplificando, el contexto permite conocer el estado actual en que se encuentra la aplicación o el objeto, y da a entender a los nuevos objetos la situación en la que se encuentran al ser creados. Se puede obtener el contexto mediante los métodos *getApplicationContext()* (*applicationContext* en Kotlin), *getContext()*, *getBaseContext()* (*baseContext* en Kotlin) o *this* (cuando la clase extiende desde *Context*, como las clases *AppCompatActivity*, *Application*, *Activity*, *Service* e *IntentService*).

El uso más común de *Context* se podría resumir de la siguiente forma:

- Crear nuevos objetos (vistas, *activities*, *adapters*, *listeners*, *providers*, etc).
- Acceso a componentes (*intents*, *ActionBar*, etc).
- Acceso a recursos (imágenes, *strings*, ficheros, etc).

Para entender el contexto, es necesario tener claro lo que significa, ya que Android respeta el significado, para ello fíjate en la definición de contexto ofrecida por la RAE[17], concretamente los puntos 1 y 2.

contexto

Del lat. *contextus.*

1. m. Entorno lingüístico del que depende el sentido de una palabra, frase o fragmento determinados.

2. m. Entorno físico o de situación, político, histórico, cultural o de cualquier otra índole, en el que se considera un hecho.

3. m. desus. Trabazón, composición o contenido de una historia o discurso.

4. m. desus. Enredo, maraña o unión de cosas que se enlazan y entretejen.

Por ejemplo, si se habla de recorrer una distancia, pongamos Alicante-Valencia, en la actualidad, en 2022, se estaría hablando de unos 160 kilómetros en coche, lo que sería 1 hora y 50 minutos aproximadamente. Pero si este mismo ejemplo, se contextualiza en el año 1492, ya no se viajaría en coche, digamos a caballo, por lo que el tiempo de viaje sería de unas 12 horas aproximadamente, siempre y cuando no descansase el caballo.

16 *Context* (https://developer.android.com/reference/android/content/Context)
17 RAE (https://dle.rae.es/contexto)