Introducción a Android

Los primeros dispositivos móviles sólo se podían programar en **bajo nivel**. Esto obligaba a los programadores a conocer íntimamente el hardware sobre el que trabajaban. Esto con el tiempo fue evolucionando, permitiendo a los programadores abstraerse por completo del hardware. De esta época data por ejemplo el **Symbian**, un sistema operativo para móviles que contaba con la posibilidad de instalar aplicaciones. Aun así, este tipo de plataformas requería utilizar código C/C++ razonablemente complejo que a demás hacía uso de librerías de bajo nivel.

Otro problema era que tampoco había un único sistema operativo, por lo que era casi imposible estar al tanto de todos ellos, los diferentes tipos de hardware, etc. La primera solución fue dotar a los sistemas operativos de la capacidad de dar soporte a **Java ME**. Al tratarse de un lenguaje multiplataforma, fue posible desarrollar aplicaciones abstrayéndose del hardware, aunque a costa de estar limitado por la propia máquina virtual de Java.

Esto motivo la aparición de Android, cuya primera versión data del 2009 coincidiendo con la proliferación de los *smartphones* y las pantallas táctiles. Desde entonces han ido apareciendo diferentes versiones, desde la **1.5 Cupcake** (basada en el núcleo de Linux 2.6.27) hasta las actuales, orientadas a tablets, televisiones, etc. Cada versión del Sistema Operativo tenía un nombre inspirado en la repostería y que seguía un orden alfabético, al menos hasta a versión **1.9 Pie**. A partir de entonces las versiones pasan a llamarse Android 10, Android 11, etc.

Versión	Nombre	Versión	Nombre
1.0	A – Apple Pie	4.1/4.2/4.3	J – Jelly Bean
1.1	B – Banana Bread	4.4	K – Kit Kat
1.5	C – Cupcake	5.0/5.1	L – Lollipop
1.6	D – Donut	6.0/6.1	M – Marshmallow
2.0/2.1	E – Éclair	7.0/7.1.2	N – Nougat
2.2	F – Froyo	8.0	O – Oreo
2.3	G - Gingerbread	9.0	P – Pie
3.0/3.1/3.2	H – Honeycomb	10.0	Android 10
4.0	I – Cream Sandwich	11.0	Android 11

Android es un Sistema Operativo de **código abierto** para dispositivos móviles. Se programa principalmente en **Java** o en **Kotlin**, y su núcleo está basado en **Linux**. Tanto el S.O. como la plataforma de desarrollo están liberados bajo la licencia de **Apache**. Esta licencia permite a los desarrolladores añadir sus propias extensiones propietarias sin tener que ponerlas en manos de la comunidad de *software libre*. A demás, el hecho de ser *open source* le otorga varias ventajas:

- Una gran comunidad de desarrollo, con APIs completas y documentación libre.
- Se puede desarrollar en cualquier plataforma: Linux, Mac, Windows…
- Cualquier fabricante puede desarrollar un dispositivo que funcione con Android adaptando o extendiendo el sistema para adaptarlo a sus necesidades.
- Se ahorra en el desarrollo de un S.O., API para gráficos, acceso al hardware, etc.

Android está formado por los siguientes componentes:

- Núcleo basado en Linux para el manejo de memoria, procesos y hardware.
- Bibliotecas open source para el desarrollo de aplicaciones, como SQLite, Webkit, OpenGL, etc.
- Entorno de ejecución para las aplicaciones Android.
- Un framework de desarrollo que pone a disposición de las aplicaciones los servicios del móvil, como geolocalización, sensores, proveedores de contenido, etc.
- Un SDK (kit de desarrollo) que incluye el Android Studio, emuladores, ejemplos, …
- Interfaz adaptado a las pantallas táctiles.
- Apps preinstaladas que hacen al S.O. útil para el usuario desde el primer momento.
- **Google Play**, quizás la pieza más importante, pues permite a los desarrolladores publicar sus propias apps tanto gratuitas como de pago.

Versiones de API

Como hemos mencionado antes, existen diferentes versiones de la plataforma Android cada una de ellas identificada por un **nombre** y una **versión**. Adicionalmente todas ellas tienen un código numérico llamado código API o **nivel de API**. A la hora de desarrollar una app es imprescindible prestar atención al nivel de API. Si desarrollamos una app para un nivel de API de 29 (Android 10) corremos el riesgo de que no funcione o ni siquiera se instale para un móvil con un nivel de API inferior.

Nombre	Versión	Nivel API
Cupcake	1.5	3
Froyo	2.2.X	8
Kit Kat	4.4	19
Lollipop	5.0	21
Android 10	10.0	29

Práctica 0

Antes de empezar en serio a programar apps para móviles, entender lo que es el ciclo de vida de una *activity*, o un *widget*, o cómo se accede a la geolocalización; vamos a hacer una práctica totalmente guiada. Vamos a implementar un sencillo *Hola Mundo* en Android Studio, lo vamos a instalarlo en un dispositivo móvil (o a emularlo).

Una vez tengas instalado el Android Studio, abre el documento **00 – Práctica Cero [Java]**. Sigue los pasos que allí te indican para completar la práctica. No importa que no entiendas lo que estás haciendo, lo que importa es que vayas familiarizándote con el IDE y en cómo se programa en Java para Android. Basta con que te funcione.

Práctica 1

Intenta hacer la misma práctica, pero en Kotlin. Lo tienes todo bien explicado en el documento **00 – Práctica Cero [Kotlin]**. Fíjate que podrás aprovechar la mayoría de lo que has hecho en la práctica anterior. De nuevo, la idea es que vayas familiarizándote con el IDE y con Kotlin. Basta con que te funcione.

Por cierto: no seas burro y no abras dos Android Studio a la vez.

Práctica 2

Realiza la práctica propuesta en el documento **01 – Números Primos [Java]**. Debería de ser sencilla de realizar con lo que ya sabes.

Práctica 3

Realiza la práctica propuesta en el documento **01 – Números Primos [Kotlin]**. Debería de ser sencilla de realizar con lo que ya sabes.

Nota: Si terminas las practicas antes que el resto de la clase, sigue adelante con la siguiente o investiga por tu cuenta. ¡No te quedes ahí parado! Tienes una caja de juguetes con apps que han hecho otros alumnos de otros años, a lo mejor te sirven de inspiración.

El Archivo Manifest

Todo proyecto Android contiene un archivo principal de configuración llamado **AndroidManifest.xml**. No hemos hablado de él en las prácticas anteriores, pero va siendo hora. Este archivo establece una serie de metadatos, estructura, componentes y requisitos de tu app. Se debe de añadir un nodo al fichero por cada uno de los componentes de la app: actividades, servicios, proveedores de contenidos, … También se indican aquí otras cosas como el icono de la app, la etiqueta, etc.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.buttonandtextview">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

El elemento raíz es <manifest>, que indica en su atributo package el nombre del paquete del proyecto donde vamos a dejar el código fuente. De él cuelga un único nodo <application> con la metainformación de la aplicación. Adicionalmente, aquí se especificarán cada una de las <activity> de nuestra app. A su vez, cada activity dispondrá de uno o varios <intent-filter> para especificar los **intents** a los que puede responder (lo veremos más adelante).

Normalmente no es necesario editar a mano el **AndroidManifest.xml**. Puede ocurrir que queramos cambiar el icono, o darle ciertos permisos a la app; pero eso suele requerir cambios menores. A demás, para crear una activity nueva, es mejor dejar que se encargue el propio Android Studio de crearla y registrarla aquí.

Práctica 4

Cambia el icono de una de las apps que has hecho hasta ahora. Android Studio dispone de herramientas para facilitarte la tarea. Si quieres usar una imagen totalmente nueva para el icono, tendrás que incluirla en la carpeta *drawable* de recursos. Investiga un poco.

El ciclo de ejecución de una app

Esto es tan complicado como importante. Tú cuando creas un programa en cualquier otro entorno, ese programa tiene un control absoluto sobre su propio ciclo de ejecución. Para entendernos, se arranca, se para y se pone en pausa cuando quiere, sin más.

Bien. Eso no pasa en Android.

Las apps en Android tienen **muy poco control** sobre su ciclo de ejecución. Deben estar atentas a sus cambios de estado y a reaccionar en consecuencia. A demás, las apps están preparadas para la finalización repentina de su ejecución. Piensa que tu app estará funcionando en un **dispositivo móvil**, por lo tanto, el usuario puede encender otra app en cualquier momento, poner una en segundo plano, volver a ella, recibir una llamada, etc.

Por defecto, cada aplicación Android se ejecuta **en su propio proceso** (ver Procesos en PSP), cada uno con su propia instancia en la máquina virtual de Android (Dalvik o ART). La forma que tiene de **seleccionar** el proceso más adecuado para ponerse en ejecución es muy especial. Android está pensado para hacer caso prioritariamente a las interacciones del usuario excluyendo todo lo demás, así que a veces hay apps y procesos en segundo plano que dejan de ejecutarse, incluso sin previo aviso, para que se puedan liberar recursos para las apps con las que el usuario está trabajando.

El orden en el que los procesos de las apps **son detenidos** viene determinado por su prioridad, que se establece en base al estado en el que se encuentre su activity de mayor prioridad.

Prioridad	Proceso	
Crítica	**Activo**	Aquellos con los que el usuario está interactuando en este momento. Android liberará recursos para asegurarse de que responden sin latencia. Sólo se los detiene si no hay más remedio.
Alta	**Visible**	Visible pero inactivo, ya sea porque la interfaz está oculta o no se está interaccionando con ellos. Puede que haya otra interfaz por encima, haya aparecido un diálogo o no ocupa toda la pantalla.
	De Servicio	Los servicios permiten que haya una ejecución de código sin interfaz con el usuario.
Baja	**Inactivos**	Actividades que ni son visibles ni están realizando tareas, y que tampoco ejecutan ningún servicio. El orden en el que se detendrán depende del tiempo que lleven inactivos.
	Vacíos	Un proceso terminado pero cacheado en memoria por Android, no vaya a ser que el usuario quiera volver a lanzarlo. Por supuesto, son de prioridad mínima.

Recursos

Todo aquello que no es código fuente (Java o Kotlin) en una app está separado en otra parte. Esto incluye a casi todo: imágenes, sonidos, textos, etc. incluyendo a los **layouts** que conforman las interfaces gráficas. Esto permite una flexibilidad enorme. Si mantenemos por separado textos en varios idiomas, al instalar nuestra app en un móvil se mostrará en el idioma por defecto del terminal. Si ponemos la misma imagen en diferentes resoluciones, se mostrará la que mejor se adecúe a la pantalla del móvil. De la misma forma, es posible diseñar layouts que se adapten a los diferentes tamaños de pantalla, o incluso hacer layouts diferentes para móviles y para tablets.

Todos los **recursos** de nuestra app se almacenan dentro de la carpeta **res** del proyecto. Ya has tenido oportunidad de trastear con ella un poco el las prácticas anteriores. Las subcarpetas que hay son:

- **Values**: para cadenas de texto, listas y valores simples.
- **Drawable y mipmap**: imágenes y recursos gráficos.
- **Layout**: interfaces para las actividades.
- **Menu**: definición de menús de opciones.
- **Raw**: otros recursos en formato crudo, no procesados ni optimizados.

Cuando se compila la app, estos recursos se incluyen en el paquete APK o App Bundle que se instalará en nuestro dispositivo. Desde nuestro proyecto podemos acceder a los recursos de forma automática mediante **una clase llamada R**. De esta forma podemos acceder a un texto guardado en strings.xml de la subcarpeta values desde nuestra activity. Un ejemplo más concreto podría ser cuando le indicamos a una activity el layout que tiene que cargar en su método **onCreate ().**

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Actividades

A groso modo, una **activity** es cada una de las **pantallas** que forman nuestra app. Cada una de ellas tiene un **layout** relacionado. Cuando arrancamos una app se ejecuta la actividad configurada como principal en el **AndroidManifest.xml**, de haber varias, y podremos ir de una actividad a otra según nos convenga.

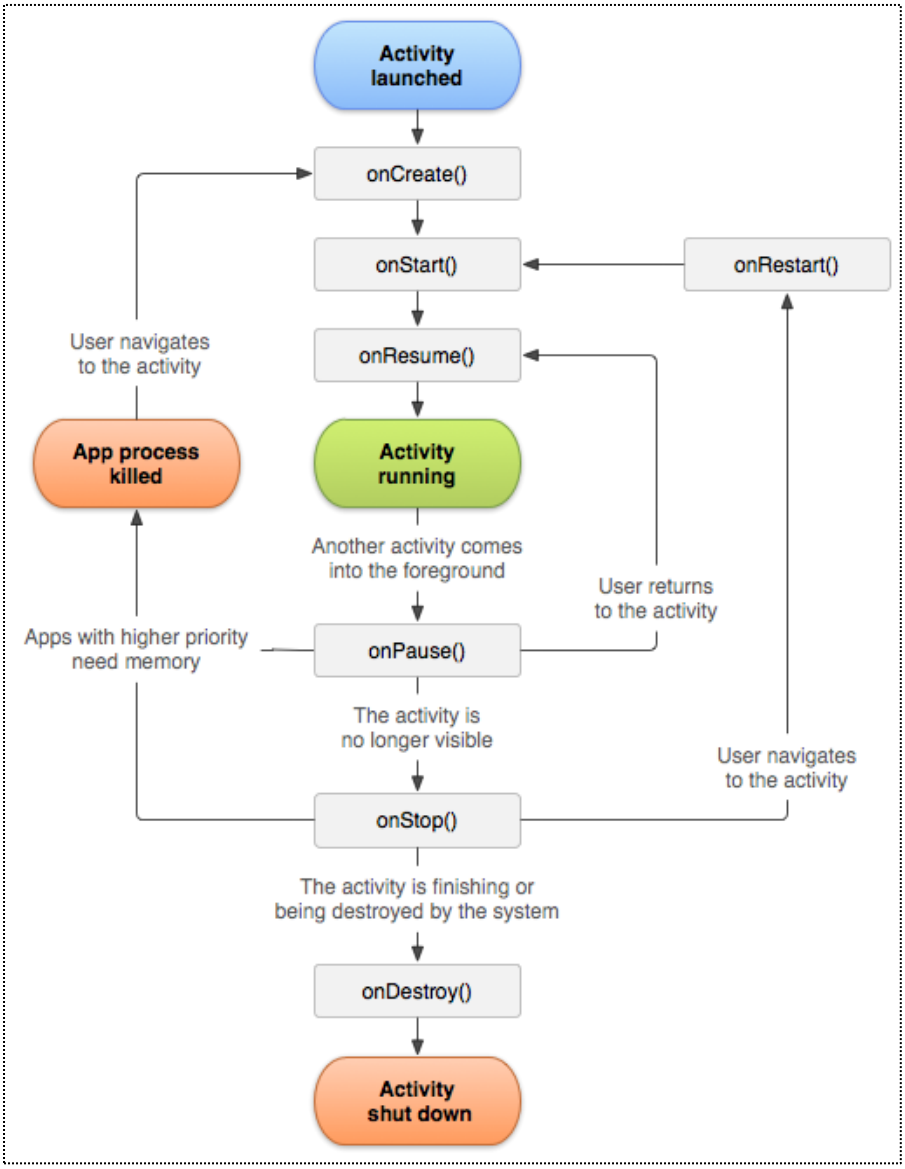
En Android **no existe** el método void main como en Java. Las activity tienen un ciclo de ejecución que hay que conocer. Aunque para la mayoría de las apps sencillas nos basta con poner código en el método **onCreate ()**, hay que entender cómo funciona todo esto.

Toda actividad dentro de una app tiene un **estado**. Este estado servirá para determinar su prioridad en el contexto de su app padre. Recuerda que la prioridad de la app depende de la prioridad de su activity de mayor prioridad. El estado viene determinado por su posición en la **pila de actividades**. Esta es una pila LIFO (Last-In-First-Out) en la que se encuentran todas las activity actualmente en ejecución. Cuando empiezas una nueva activity, la que se estaba mostrando en ese momento pasa al tope de la pila. Si le damos a 'volver' o cerramos esa activity, la activity del tope de la pila saldrá de ella y pasará a ser la activa.

Estados	
Activa	La activity se está ejecutando en este momento, es visible, tiene el foco y el usuario puede interaccionar con ella. Android intentará por todos los medios mantener esta activity en ejecución.
En pausa	La activity está activa, pero sin foco. Suele ocurrir que su interfaz está por debajo de otra, o que no esté en pantalla completa. No puede recibir interacciones del usuario.
Detenida	La activity no es visible, pero permanece en memoria con todos sus datos. La pega es que puede ser **eliminada** de memoria en cualquier momento. Normalmente, cuando una activity pasa a detenida, se suelen almacenar los datos y el estado de la interfaz.
Inactiva	La activity pasa a inactiva cuando se ha terminado su ejecución o todavía no se ha iniciado. Se han sacado de la pila y deben de ser lanzadas de nuevo.

Todo esto es transparente para el usuario, que se limita a mover el dedo sobre la pantalla del móvil sin más. Pero los **programadores** podemos gestionar estos cambios de estado mediante eventos. Para ello, simplemente sobrecargamos el método correspondiente de la clase Activity.

A continuación, tienes un esquema de los **estados**, los **eventos** y la **secuencia** en la que se producen. Por ahora no es necesario tenerlo en cuenta, pero sí a futuro.



Eventos del ciclo de vida de una activity:

- **onCreate ()** – Se añade aquí el código necesario para inicializar el interfaz.
- **onStart ()** – Se llama cuando el la activity pasa a estado visible.
- **onResume ()** – Se llama cuando la activity pasa a estado activo. Se suelen poner aquí los hilos (Threads) que mueven muñequitos de los videojuegos (por ejemplo).
- **onPause ()** – Se llama al pausar una actividad. Todo lo que haga onResume () tiene que deshacerse en onPause ().
- **onStop ()** – Se llama cuando una actividad deja de ser visible. Todo lo que se haga en onStart () debería deshacerse en onStop ().
- **onRestart ()** – Se llama cuando una actividad vuelve a ser visible después de haber estado no visible.
- **onDestroy ()** – Se llama cuando se va a destruir la actividad. Debería deshacer todo lo hecho en el onCreate ().

Práctica 5

Realiza la práctica propuesta en el documento **02 – Calculador [Java]**. Debería de ser sencilla de realizar con lo que ya sabes.

Práctica 6

Realiza la práctica propuesta en el documento **02 – Calculador [Kotlin]**. Debería de ser sencilla de realizar con lo que ya sabes.