Entorno de trabajo. Instalación y primer programa con Android Studio.

# 1.- Módulos para el desarrollo de aplicaciones. Instalación de Android Studio.

Una vez sabemos las características del entorno de ejecución de una aplicación **Android**, ¿Qué bibliotecas adicionales hay que instalar para poder desarrollar con **Android**? ¿Es suficiente con el **SDK** clásico? ¿Es necesario o aconsejable tener también un **IDE**?

* **Para compilar un programa Android**: se necesita tener mínimo el SDK. A partir de ahí se pueden descargar e instalar el resto de herramientas (emuladores, entornos de desarrollo, depurador, etc.).
  + En este caso particular, puedes optar por instalar el **SDK** junto con otros elementos, o bien utilizar algún otro entorno compatible que traiga todas las herramientas necesarias incorporadas. Para este curso, usaremos el entorno **Android Studio**, con los paquetes necesarios para trabajar con **Android**.
* **Del mismo modo que se haría con Android**, podría hacerse con cualquier otra plataforma de desarrollo con la que se hubiera decidido trabajar (Java ME, iOS, Windows Phone, etc.) En cada caso, serían necesarias unas herramientas específicas (compiladores, editores, depuradores, emuladores, etc.).

Entorno integrado de desarrollo: Instalación de Android Studio.

Para poder desarrollar una aplicación **Android**, necesitarás un IDE (editor, copmilador, depurador, emulador, etc.). En nuestro caso, se ha decidido que trabajaremos con **Android Studio**. Esta será la herramienta que utilizarás a partir de ahora y durante el resto del curso para el desarrollo de aplicaciones sobre dispositivos móviles.

Para la instalación de **Android Studio**, nos remitimos a la [página oficial de Android Studio[Acceder a página oficial de la instalación de Android estudio (se abre en una nueva ventana)](https://developer.android.com/studio/install.html)](https://developer.android.com/studio/install.html).

En ella podrás visualizar un video descriptivo de todo el proceso de instalación sea cual sea tu sistema operativo, puesto que, al ser una aplicación multiplataforma, está completamente testada y optimizada para los sistemas operativos más habituales (**Windows**, **Mac** y **Linux**)

En cuanto a los **requisitos del sistema**, necesitarás los especificados en la página oficial. Aunque **será deseable disponer de los mejores recursos hardware de los que podamos disponer** ya que el proceso de carga del emulador será algo tedioso por muy elevadas que sean las prestaciones del sistema.

Captura de pantalla de un celular

Descripción generada automáticamente

**Debes conocer**: es importante que conozcas los requisitos mínimos que Android Studio necesita para ejecutarse correctamente.

# 2.- Integración en el entorno de desarrollo.

Una vez se ha descargado e instalado **Android Studio**, debemos ver qué pasos debemos seguir para desarrollar, probar y ejecutar una mínima aplicación móvil con esta herramienta.

**Recomendación**: en caso de no contar con los permisos necesarios para instalar la aplicación en tu ordenador, debes conseguirlos.

Plantéate usar una máquina virtual como las que has utilizados en otros módulos del curso. Pero ten en cuenta que para poder probar tus aplicaciones, Android Studio usa lo que se llama un virtualizador de dispositivos Android (AVD). Este depende de la configuración de tu SW de virtualización (VirtualBox, HyperV,...), la configuración de tu BIOS e incluso tu microprocesador y hay muchos casos en los que no se comporta de forma adecuada.

Te recomendamos por tanto, que uses una máquina real para evitar problemas con los emuladores y poder incluso instalar otros diferentes a los que te ofrece Android Studio de un modo más sencillo.

## 2.1.- Creación de un proyecto.

Para describir este apartado, usaremos la referencia proporcionada desde la página de [Cómo crear un proyecto | Android Studio | Android Developers](https://developer.android.com/studio/projects/create-project?hl=es-419)

Un proyecto de **Android Studio** contiene uno o más módulos que mantienen tu código organizado en unidades de funcionalidad discretas.

**Android Studio** facilita la creación de apps de Android para varios factores de forma, como teléfonos y tablets, **Wear OS** , Android TV y **Android Automotive**. El asistente **New Project** te permite elegir los formatos para tu app y completar la estructura del proyecto con todo lo que necesitas para comenzar. Revisa y sigue los siguientes pasos para crear un proyecto nuevo.

**Pasos a seguir**:

1. **Crear un nuevo proyecto**.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

1. **Agregar una actividad**.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. **Configuración del proyecto**. Introduce los siguientes datos de configuración:
   1. **Nombre de la aplicación**: Hola mundo
   2. **Nombre del paquete**: com.miempresa.holamundo
   3. **Ubicación para guardar el proyecto**: por defecto o la que elijas
   4. **Lenguage**: Java
   5. **Nivel API mínimo**: escoge aquel que permita que nuestra app se ejecute en valores cercanos el 100% de los dispositivos
   6. **No marcar** la opción **“Use Legacy android.support libraries”**
2. Seleccionar factores de forma y el nivel de API.
   1. Este paso no lo realizamos ahora, pero lo comentamos para verlo más adelante en el curso.
   2. **Los formatos seleccionados** (teléfonos, tablets, War OS, Android TV y Android Automotive) **se convierten en los módulos de la app** dentro del proyecto. **Para cada formato, también puedes elegir el nivel de API para esa app**.
   3. Asimismo, **podremos seleccionar las versiones mínimas** para los distintos factores de forma. Esta selección hará que nuestra aplicación sea compatible con más o menos versiones de Android, tal y como se muestra en esta tabla.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

* 1. **Nota**: para eliminar módulos añadidos, nos colocamos encima de la carpeta del módulo añadido y hacemos clic con el botón derecho “Open Module Settings” para eliminarlo con el “.”. Después, podemos suprimir la carpeta.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Cada factor de forma será un módulo en nuestra app, como se puede ver en la imagen anterior (en este caso, hemos supuesto que hemos elegido los factores de forma Phone and Tablet y TV).

1. **Actividad principal**.

Una actividad (**activity**) es un **componente** de la app que contiene **una pantalla** con la que los usuarios pueden interactuar para realizar una acción, como marcar un nº telefónico, tomar una foto, enviar un email o ver un mapa.

**A cada actividad se le asigna una ventana** en la que se puede dibujar su interfaz de usuario. La ventana generalmente abarca toda la pantalla, pero en ocasiones puede ser más pequeñas y quedar “flotando” encima de otras ventanas.

Una app generalmente tiene **varias actividades vinculadas de forma flexible entre sí**.

Por ahora nuestro proyecto sólo tiene una actividad que es la principal representada por:

* **La clase MainActivity** (con fichero de código fuente MainActivity.java)
* **La disposición o layout activity\_main** (almacenada en el fichero XML: activity\_main.xml) que contiene la información necesaria para dibujar en pantalla la interfaz gráfica de la actividad principal

**Nota**: no es obligatorio, sin embargo, por ahora, la primera actividad que crearemos será la actividad principal de la aplicación. Mantendremos el nombre **MainActivity** con fichero de interfaz **activity\_main**, lo cual ayudará a identificar y organizar las partes de la aplicación. Por otro lado, comentar que **la actividad principal no tiene por qué ser la primera actividad que se abre en la aplicación**. Puede suceder, que antes de abrir la actividad principal, se abra lo que se conoce como una pantalla de bienvenida (splash screen) que ayuda a mejorar la experiencia de usuario. Esto será habitual en aplicaciones más trabajadas o en fase de producción.

Para ver la relación existente entre la clase **MainActivity** y su disposición (**layout**) podemos pulsar Ctrl+BIzq (Botón izquierdo del ratón) sobre el nombre de la clase MainActivity en la línea de código *"public class MainActivity*" y nos llevará al fichero XML asociado. Asimismo si pulsamos Ctrl+BIzq sobre el nombre de una clase, el entorno de desarrollo Android Studio nos llevará a su definición; puedes probar pulsando sobre *Bundle* (el tipo del parámetro del método onCreate). En el caso de MainActivity no nos llevó a su definición ya que estamos en ella (en el fichero *MainActivity.java* donde se define la propia clase); en vez de eso nos mostró los sitios donde se usa esta clase (uno de ellos es el fichero XML de la disposición asociada).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. **Desarrollar tu app**. Android Studio **crea la estructura predeterminada de tu proyecto** y se abre el entorno de desarrollo y se abre el entorno de desarrollo. **Si tu app admite más de un formato, Android Studio crea una carpeta de módulos** con archivos de origen completos para cada uno de ellos, como se aprecia en la siguiente figura, donde se puede apreciar que:

Interfaz de usuario gráfica

Descripción generada automáticamente

* 1. **En la parte superior**, hay un menú y/o barras de herramientas
  2. **En el panel lateral izquierdo**, hay un navegador de la estructura de ficheros del proyecto.
  3. **En la parte central**, tenemos un panel con una paleta de elementos incorporables en una emulación de la interfaz visual de nuestra aplicación.
  4. En el panel lateral derecho disponemos de un árbol de componentes de la interfaz visual, así como un descriptor de propiedades de cada elemento incorporado.

**Para saber más**: sobre la estructura del proyecto y los tipos de módulos de Android: [Descripción general de proyectos | Android Studio | Android Developers](https://developer.android.com/studio/projects/index.html?hl=es-419), y sobre [Cómo agregar un módulo para un dispositivo nuevo | Android Studio | Android Developers](https://developer.android.com/studio/projects/add-app-module.html?hl=es-419)

## 2.2.- Escritura de código.

Lo más habitual es partir de una aplicación previa básica que a modo de esqueleto permita agregar las particularidades necesarias para dar respuesta a las necesidades a cubrir por nuestra aplicación.

Concretamente, vamos a trabajar sobre la plantilla **"Empty Activity"** que tiene los elementos estrictamente necesarios para poder ejecutar la aplicación generada. Además, es habitual también, en los primeros pasos con Android Studio, mantener el nombre de actividad que pone por defecto (**MainActivity**), de forma que cuando arranque la aplicación, esta será la actividad (**activity**) que cargará en primer lugar.

Todas las actividades que vayamos incorporando en nuestra aplicación quedarán reflejadas en el fichero ***AndroidManifest.xml*** que veremos posteriormente. Además dicho fichero permitirá establecer cúal será la actividad con la que arrancará la aplicación.

*Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. Normalmente, una actividad es “principal” cuando quieres que sea la primera que se presente al usuario al iniciar la aplicación*. *Cada actividad puede a su vez iniciar otra actividad para poder realizar diferentes acciones. Cada vez que se inicia una actividad nueva, se detiene la actividad anterior, pero el sistema conserva la actividad en una pila (la "pila de actividades").*

*Cuando se inicia una actividad nueva, se la incluye en la pila de actividades y capta el foco del usuario. Cuando el usuario termina de interactuar con la actividad actual y presiona el botón Atrás, se quita de la pila (y se destruye) y se reanuda la actividad anterior.*

De forma un poco simple, podremos asociar el concepto de **Activity** al de "pantalla" o ventana de la aplicación, de forma que todas las diferentes pantallas que pudiéramos ver en una aplicación tendrán asociadas una actividad. **Las actividades estarán compuestas por una parte lógica y una parte gráfica**.

* **Parte lógica**: se corresponde con un archivo Java, que contiene la clase para poder interactuar desde esa actividad, a través de los diferentes métodos de que disponga, e incluso apoyándose en otros ficheros .java que tengan clases que también interactúen entre sí.
* **Parte gráfica (interfaz)**: se corresponde con un fichero ***.xml*** en el que se define la disposición (**layout**) de los elementos que se mostrarán

Más adelante, cuando planteemos alguna aplicación con interfaz “programática”, que será posible incorporar los elementos gráficos directamente en el fichero **.java**, pero en general, esto suele ser bastante más costoso en esfuerzo y tiempo.

Al crear nuestra primera aplicación, podemos ver ambas facetas. Para ello, completamos la creación de nuestro primer proyecto (llamado *HolaMundo*), y **vemos que a través del navegador de proyectos ubicado en el marco (*frame*) de la izquierda se crean 2 ficheros relacionados entre sí**:

* **MainActivity**, es la clase Java definida en el fichero *MainActivity.java*, y que contendrá las especificaciones lógicas de la actividad principal y de momento única, de nuestra aplicación.
* ***activity\_main.xml*** contendrá las especificaciones de apariencia o diseño de la interfaz de esa actividad.

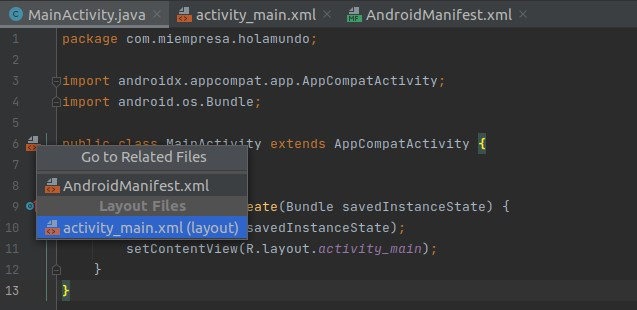
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

### 2.2.1.- MainActivity.

**Contenido de MainActivity**:

Para ver los archivos XML relacionados con la clase Java principal de la aplicación, Debes pulsar en el icono "Related XML file" tal y cómo se muestra en la imagen para ver los ficheros XML que están relacionados con la clase *MainActivity*: en este caso, *activity\_main.xml* y *AndroidManifest.xml*.



**Código del MainActivity**:

Texto

Descripción generada automáticamente

* Con las líneas 3 y 4 incorporamos clases necesarias para nuestra app, las cuales están en otros paquetes.
* En la línea 6 definimos la clase llamada "MainActivity", la cual hacemos que herede de la clase AppCompatActivity. Esta clase dispone de las definiciones necesarias para cargar una actividad.
* En la línea 9, **se sobreescribe el método “onCreate”**. **Todas las actividades (activities) deben llevar el método "onCreate"**, el cual hará que inicie la actividad. Se debe pasar como parámetro un **objeto de la clase *Bundle*, el cual permitirá inicializar el estado de la actividad**. Si no programamos este método la actividad no se iniciará.
  + Lo primero que tendremos que hacer es llamar al método onCreate de la clase padre pasando el objeto Bundle anterior.
  + A continuación, a través de la línea de código: setContentView(R.Layout.acivity\_main) conseguiremos enlazar la parte lógica con la parte gráfica, pasando como parámetro el nombre que recibe dentro del registro de recursos que tiene Android y que para las actividades tiene prefijo “R.layout”, como por ejemplo sucede en nuestro caso.

**Debes conocer**: En la línea 3, si pulsas Ctrl+BIzq sobre AppCompatActivity se abre el fichero de código fuente donde se define el paquete en el que se incuye dicha clase. Idem para la clase Bundle de la sentencia import de la línea 4. Observa brevemente la descripción de esas clases en el comentario (/\*\* ... \*/) que aparece encima de la definición de la clase.

### 2.2.2.- activity\_main.xml.

Texto

Descripción generada automáticamente

* En la línea 1, ese primer elemento del fichero XML, define la versión de XML y la codificación de caracteres utilizada. Habitualmente, mantendremos estos valores.
* En la línea 2, la etiqueta **<Android.support.constraint.ConstraintLayout…>**, determina el tipo de disposición (**layout**) que queremos que se establezca, e incorpora una serie de atributos correspondientes a los espacios de nombres indicando unas URI’s para identificarlos.
  + Atributos para la anchura y altura de la disposición (layout).
  + Especialización del contexto de esa disposición (layout) que servirá para recordar qué actividad (**activity**) la asociará.
* Por otro lado, tenemos una etiqueta **<TextView...>** , similar al control *Label* de Java o *label* de html. Dispondrá de atributos:
  + para especificar anchura y altura,
  + el texto contenido y,
  + la forma en que se situará este control dentro de la disposición (**layout**).

**Para saber más**: sobre el acceso a recursos de Android Studio [Descripción general de los recursos de las apps | Desarrolladores de Android | Android Developers](https://developer.android.com/guide/topics/resources/providing-resources?hl=es-419#Accessing)

## 2.3.- Compilación.

Para compilar el proyecto, basta con ir a **Build > Make Project** y veremos que se empieza a ejecutar Gradle.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para compilar y además ejecutar tu app, bastará con hacer clic en el botón **Run.**

Android Studio compilará la app con **Gradle**, y a continuación, solicitará que **seleccionemos un destino de implementación** (un emulador o un dispositivo conectado) y, luego, cargará la app en dicho destino. **Si no tenemos ningún dispositivo preparado** para poder emular la aplicación, Android Studio mostrará un mensaje similar a este: ***Error running app: No target device found****.*

Podremos personalizar parte de este comportamiento predeterminado (p.e: la selección de un destino de implementación automático). Veremos en siguientes apartados como ejecutar aplicaciones en el emulador y como ejecutarlas en un dispositivo físico.

**Debes conocer**: Además, puedes ejecutar tu app en el modo de depuración (y de paso, hacer uso de los puntos de interrupción en el código, examinar variables y evaluar expresiones en tiempo de ejecución). Para ello, haz clic en el botón de **Debug.** [Cómo depurar tu app | Android Studio | Android Developers](https://developer.android.com/studio/debug/index.html?hl=es-419)

## 2.4.- Emuladores.

Normalmente, los IDE suelen incorporar uno o varios emuladores que te servirán para poder probar las aplicaciones antes de instalarlas en el dispositivo para el cual se han programado.

Los emuladores no siempre van a coincidir en aspecto (y a veces incluso en funcionalidad) con el dispositivo real, y podrás encontrar abundantes diferencias dependiendo de, por ejemplo, el tipo de dispositivo que se emule. No es lo mismo emular un teléfono con **Windows Phone**, que un **iPhone** o un **iPad**, o un teléfono con **Android**, o un dispositivo móvil con **Java ME**.

Independientemente de la plataforma para la que vayas a desarrollar las aplicaciones (**Android, iOS, Java ME,** etc.), gran parte del proceso de depuración se realizará con el emulador, podremos ejecutar las instrucciones paso a paso, establecer puntos de ruptura (breakpoints), observar el contenido de variables, etc. Además, en muchos casos es posible que no tengas a tu disposición en ese momento el dispositivo concreto para el que deseas programar de manera que necesitarás el emulador.

**¿Con qué emulador trabajaremos nosotros?** Con la aplicación de emulación proporcionada por Android (**AVD**), pero es posible utilizar otras aplicaciones gratuitas o de pago que consiguen emulaciones tan buenas o incluso más rápidas que la AVD. Éstas, normalmente, se valen del uso de máquinas virtuales para poder funcionar.

Interfaz de usuario gráfica

Descripción generada automáticamente

Para ejecutar nuestra app en este emulador, debes preparar un **Android Virtual Device (AVD)**. Si todavía no lo hemos creado, cuando hagamos clic en Run, aparecerá un error ya que la app no encuentra un dispositivo sobre el que ejecutarse. Simplemente, debemos abrir ***Tools > Device Manager*** desde donde podremos elegir ***Create Device***. Seguiremos las instrucciones, definiendo el tipo de dispositivo que deseas emular, seleccionando la categoría (teléfono, TV, Wear, Tablet o Automotive), el modelo y la imagen del sistema (a descargar).

**Para saber más**: [Cómo crear y administrar dispositivos virtuales | Android Studio | Android Developers](https://developer.android.com/studio/run/managing-avds.html?hl=es-419), [Los mejores emuladores de Android para PC (xataka.com)](https://www.xataka.com/basics/mejores-emuladores-android-para-pc)

## 2.5.- Instalación y ejecución en un dispositivo real.

Algo que se debe tener en cuenta cuando se desarrollan aplicaciones, es que **puede variar el aspecto y funcionamiento de un emulador a otro emulador, o a un dispositivo real**, debido a las características físicas del propio dispositivo (tamaño, resolución, colores de la pantalla, tipo de mando de juegos, botones, etc.) y a la implementación de la máquina virtual de Java que tenga ese dispositivo.

La manera más sencilla de probar tu aplicación será transfiriendo el **archivo apk** generado de la aplicación a la memoria del dispositivo donde deseas realizar la prueba (a través de un cable USB, por Bluetooth, mediante una tarjeta de memoria, etc.).

En Android, disponemos de **adb** (Android Debug Bridge, es una herramienta de línea de comandos versátil que te permite comunicarte con un dispositivo). Para usarla, será necesario activar la **Depuración por USB**:

[Qué es la depuración USB en Android y para qué utilizarla - YouTube](https://www.youtube.com/watch?v=Zqg6Kl46_c4)

**Para saber más**: [Cómo ejecutar apps en un dispositivo de hardware | Android Studio | Android Developers](https://developer.android.com/studio/run/device.html?hl=es-419)

# 3.- Gestión del entorno de ejecución.

Para describir el entorno de ejecución en un sistema como Android, debemos considerar 2 aspectos importantes:

* **El ciclo de vida de una aplicación**. Fases y **administrador de aplicaciones**.
* **El modelo de estados** de una aplicación en ejecución.

## 3.1.- El ciclo de vida y el administrador de aplicaciones.

El **administrador de aplicaciones** permite mostrar todas las apps que tenemos instaladas en nuestro dispositivo y controlar de un vistazo las que se está ejecutando, desinstalarlas o detenerlas, o incluso conocer en cada momento los recursos que está consumiendo cada una de ellas. Llevará a cabo esencialmente dos funciones:

* La gestión del ciclo de vida de las aplicaciones en el dispositivo.
* El control de los estados por los que pasa la aplicación mientras resida en la memoria del dispositivo, es decir, mientras esté en ejecución.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El ciclo de vida de una aplicación Android (5 fases)

* **Descubrimiento (o localización)**: proceso por el cual un usuario localiza una aplicación Android a través de su dispositivo.
  + El gestor de aplicaciones es el que proporciona los mecanismos necesarios para realizar la elección de la aplicación que se desea descargar (mediante un cable USB, Bluetooth, conexión a Internet, etc.).
* **Instalación**: El gestor de aplicaciones de Android se encargará de controlar el proceso de instalación informando al usuario de la evolución de este proceso
* **Ejecución**: El gestor de aplicaciones de Android es también el que permite poner en ejecución la aplicación. Una vez que una aplicación está en ejecución, el gestor de aplicaciones también tendrá el control de los distintos estados por los que pueda pasar dicha aplicación dependiendo de los eventos que se vayan desencadenando durante esa ejecución
* **Actualización**: Una vez que una aplicación ha sido descargada, el gestor de aplicaciones de Android debe ser capaz de detectar si se trata de una actualización de una aplicación ya presente en el dispositivo
* **Borrado**: Si el usuario así lo desea, el gestor de aplicaciones de Android se encargará de borrar la aplicación del dispositivo. Normalmente se solicitará confirmación.

## 3.2.- Modelo de estados de una aplicación.

De manera similar a como el planificador de procesos (scheduler) en sistemas operativos se encarga de gestionar el estado de los procesos (programas en ejecución), el Administrador de Aplicaciones es también el encargado de gestionar los estados de una aplicación durante su ejecución.

Cuando se ejecuta una aplicación, un **proceso** se inicia, entrando en funcionamiento una **activity**. Es una diferencia con respecto a otros paradigmas que cargan la aplicación mediante el método ***main()***. El ciclo de vida de cualquier activity dependerá de cómo esté relacionado con otras actividades, con sus tareas y con la pila de actividades que tenga la aplicación.

Según [Android Developer[](https://developer.android.com/guide/components/activities/activity-lifecycle#asem)](https://developer.android.com/guide/components/activities/activity-lifecycle#asem), los usuarios cambian el estado de los procesos de las apps según interactúan con ellas. Los cambios en el estado del proceso van a cambiar el estado de la activity que se está ejecutando. Básicamente estarán en tres estados:

* **Reanudada (o en ejecución)**: La actividad se encuentra en el primer plano de la pantalla y tiene la atención del usuario.
* **Pausada**: otra actividad está visible por encima de esta y esa actividad es parcialmente transparente o no cubre toda la pantalla.
  + Una actividad pausada está completamente "viva" (el objeto Activity se conserva en la memoria, mantiene toda la información de estado y miembro y continúa anexado al administrador de ventanas), pero el sistema puede eliminarla en situaciones en que la memoria sea extremadamente baja.
* **Detenida**: La actividad está completamente opacada por otra actividad (ahora pasa a estar en segundo plano).
  + Una actividad detenida también permanece "viva" (el objeto [Activity](https://developer.android.com/reference/android/app/Activity.html?hl=es-419" \t "_blank" \o "Acceder a la página de referencia de la clase Activity en Developer.android.com (se abre en una nueva ventana))[[](https://developer.android.com/reference/android/app/Activity.html?hl=es-419)](https://developer.android.com/reference/android/app/Activity.html?hl=es-419" \t "_blank" \o "Acceder a la página de referencia de la clase Activity en Developer.android.com (se abre en una nueva ventana)) se conserva en memoria, mantiene toda la información de estado y miembro, pero no está anexado al administrador de ventanas).
  + Sin embargo, ya no está visible para el usuario y el sistema puede eliminarla cuando necesite memoria en alguna otra parte.

Si se pausa o se detiene una actividad, el sistema puede liberar su espacio ocupado en memoria, puede excluirla de la memoria, llamando al método **finish()**, o simplemente eliminando su proceso. Cuando se vuelve a abrir la actividad (después de haber sido eliminada o destruida), es necesario crearla nuevamente.

Cuando una actividad entra y sale de los diferentes estados que se describieron más arriba, esto se notifica a través de diferentes métodos llamados **callback.** Todos **los métodos callback** son **enlaces que puedes invalidar** para realizar las tareas correspondientes **cuando cambia el estado de la actividad**.

**NOTA**: la implementación que realices de estos métodos del ciclo de vida siempre debe llamar a la implementación de la superclase antes de realizar cualquier otra tarea.

Ejemplo: los 3 bucles del ciclo de vida de un activity

Diagrama

Descripción generada automáticamente

En conjunto, estos métodos anteriores definen el ciclo de vida completo de una actividad. Al implementar estos métodos, puedes monitorear los tres bucles anidados en el ciclo de vida de la actividad que puedes ver representados en la ilustración así como el camino que debe tomar una actividad entre estados. Los rectángulos representan los métodos callback que puedes implementar para realizar operaciones cuando la actividad cambie de estado.

* El **ciclo de vida** **completo** de una actividad transcurre entre la llamada a **onCreate()** y la llamada a **onDestroy()**.
  + **En onCraete, tu actividad debe configurar el estado “global”** (como la definición del diseño)
  + **En onDestroy()**, **tu actividad debe liberar todos los recursos** restantes.
  + **Por ejemplo**, si hay un subproceso de descarga de datos de la red en ejecución en segundo plano en tu actividad, esta podría crear ese subproceso en onCreate() y luego detenerlo en onDestroy().
* El **ciclo de vida** **visible** de una actividad transcurre entre la llamada a **onStart()** y la llamada a **onStop()**. Durante ese tiempo, el usuario puede ver la actividad en pantalla e interactuar con ella.
  + Por ejemplo, se llama a **onStop()** cuando se inicia una nueva actividad y esta ya no está visible.
  + Entre estos dos métodos, **puedes conservar los recursos necesarios para mostrarle la actividad al usuario más tarde**.
  + El sistema podría llamar a onStart() y onStop() muchas veces durante el ciclo de vida completo de la actividad, ya que la actividad pasa de ser visible a estar oculta para el usuario.
  + **No uses estos métodos para hacer acciones que sólo se realizan una vez**.
* El **ciclo de vida en** **primer plano** de una actividad transcurre entre la llamada a onResume() y la llamada a onPause(). Durante ese tiempo, la actividad se encuentra al **frente de todas las demás actividades** en la pantalla y **tiene el foco** en la interacción del usuario.
  + **Con frecuencia, una actividad puede entrar y salir de primer plano**, por ejemplo, se llama a onPause() cuando el dispositivo entre en suspensión o cuando aparece un diálogo.
  + Dado que este estado puede cambiar con frecuencia, **el código en estos métodos debe ser bastante liviano para evitar las transiciones lentas que hacen que el usuario deba esperar**.