

Tecnológico Nacional de México

Instituto Tecnológico Culiacán

Unidad V: Programación de Dispositivos Móviles

Trabajo de Evaluación:

Síntesis de Programación en Dispositivos Móviles

Asignatura: Tópicos Avanzados de Programación

Unidad V: Programación de Dispositivos Móviles

Docente: MC Jaime Arturo Félix Medina

Hora: 17:00 – 18:00

Alumno: Jaime Alonso Ruiz Lizarraga

No. De Ctrl: 19170736

Fecha: 02/12/2022

Índice

[1. Sistema operativo Android 3](#_Toc2013021169)

[2. Entornos de desarrollo Android 4](#_Toc741825921)

[3. Estructura de un proyecto Android 6](#_Toc1413940108)

[Clase R 7](#_Toc1512039814)

[fichero tipo int 7](#_Toc1891781664)

[atributo/elemento 7](#_Toc1959857331)

[Depuración 8](#_Toc1885185410)

[Registro O LOG 8](#_Toc857032387)

[4. Componentes de una aplicación Android 9](#_Toc2121474327)

[Actividades 9](#_Toc606777719)

[Concepto Clave Clase 9](#_Toc1118991258)

[Creación de una Actividad 10](#_Toc2069878585)

[Métodos importantes en una actividad 10](#_Toc247912722)

[onCreate (Bundle bundle) 10](#_Toc1608606804)

[findViewByld(int id) 10](#_Toc831796994)

[setContentView (int id) 10](#_Toc1284883609)

[finish() 11](#_Toc1549841078)

[Views y ViewGroup 11](#_Toc875664619)

[Atributos Comunes de una Vista 11](#_Toc1115732279)

[Unidades de Medida 12](#_Toc1001604335)

[Layout Básicos 12](#_Toc296054437)

[Callback y Listener 13](#_Toc1794819727)

[Visibilidad: Visible, Invisible y Gone 13](#_Toc1869737551)

[Ciclo de Vida de una Actividad 13](#_Toc1268416570)

[Estados de una Actividad 14](#_Toc1263536361)

# **1. Sistema operativo Android**

Al hablar de Android tenemos que entenderlo como sistema operativo. Android es una versión derivada de los sistemas operativos tipo Linux/Unix, competencia histórica de Windows. Y decimos que un dispositivo es Android si su sistema operativo lo es. Al realizar una aplicación Android, se debería dar como resultado un ejecutable para dichos dispositivos. También, estaremos diciendo que hemos hecho la aplicación usando Android como marco de trabajo o framework.

Cuando aprendes a programar en Java, siempre defines un punto de acceso a tu aplicación por donde comienza su ejecución, la famosa función main. Pero cuándo programas en Android, ese main ya está hecho. Al lanzar una aplicación Android, la plataforma, comienza su ejecución (en esa función main, definida de antemano), hará un montón de cosas, cargará archivos de configuración, instanciará una serie de clases y en un determinado momento, el flujo de la ejecución engarzará con nuestro código, siendo entonces invocadas las instrucciones programadas por nosotros.

Es decir, hacer una aplicación en Android, será completar un mecanismo ya dado, incorporar mi código a un sistema con un funcionamiento y diseño previos, para terminar, produciendo nuestra app. De ahí el nombre de marco de trabajo o framework.

Una aplicación en Android puede desarrollarse combinando el uso de hasta tres lenguajes: C/C++, Java y Kotlin. Una aplicación deberá ser analizada, diseñada, codificada, testeada, desplegada, mantenida y gestionada en todo su ciclo de vida. Dependiendo de su tamaño, necesitará de un arquitecto o responsable técnico, un grupo de desarrolladores, maquetadores, probadores, analistas, diseñadores y gestores.

Android se concibe para ser usado en la capa de presentación, como el cliente, la vista o el front-end, término guiri este último. Es decir, como norma se empleará para:

1. Representar la interfaz de usuario

2. Ofrecer al usuario las operaciones que este puede realizar

3. Realizar operaciones sencillas de cálculo y almacenamiento

4. Obtener y representar datos de un servidor con el que intercambiamos información

Normalmente, una aplicación moderna (WhatsApp, Facebook, Metro de Madrid, etc.) está distribuida generalmente en tres capas: cliente, servidor y base de datos, presentando esta arquitectura tipo: De izquierda a derecha: el dispositivo Android de turno, el servidor de aplicaciones y la base de datos. Todos estos elementos juntos en acción constituyen el cuerpo y la estructura habitual de una aplicación moderna. Cabe destacar que, como clientes a alternativos a Android, se emplea IOS, páginas web y aplicaciones híbridas.

# **2. Entornos de desarrollo Android**

Para el desarrollo de aplicaciones Android necesitamos descargar Android Studio. Es el entorno integrado de desarrollo (IDE, en inglés), que se utiliza para programar nuestras apps.

Antaño, era posible programar para Android desde Eclipse entorno por excelencia para desarrolladores Java, que además permitía desplegar simultáneamente tu app y el servidor, pero desde abril de 2016, se ha hecho obligado el empleo de esta herramienta. Han sido años difíciles para lograr una versión madura del IDE, pero finalmente, desde la versión 3, parece lograda cierta estabilidad. Este IDE, trae consigo todo lo necesario para poder construir nuestras aplicaciones. Los elementos destacados son:

* JDK En su versión 8, contienen todo el API estándar de Java que necesitaríamos para hacer una app de Java (javac, javadoc, las clases como File, String, etc.) Por problemas de compatibilidad con el SDK, para usar características de la versión 8 de Java necesitaremos restringir su funcionamiento al API 24, 25 o incluso 26 del SDK.
* SDK Como hemos dicho antes, son todas las clases que nos dan ya en Android. Es decir, además de las clases de Java, comentadas en el punto anterior, quedarán a nuestra disposición clases como Button, Activity o ActionBar, que iremos empleando en la creación de nuestros ejemplos.
* SDK Manager Herramienta que permite descargarnos a nuestro equipo distintas versiones del SDK o complementos necesarios para el entorno, actualizaciones, etc. Será útil, si, por ejemplo, trabajamos con la fuente de una aplicación hecha con la API 21 y en nuestro equipo sólo tenemos la API 28.
* Gradle Herramienta que gestiona el ciclo de vida de mi aplicación. Desde la gestión de dependencias, compilación, testeo, empaquetado y despliegue. Análogo a Maven, para los que vienen de entornos Java modernos, esta fue desarrollada por la gente de Google.
* AVD Android Virtual Device, es una extensión del IDE que nos permite definir dispositivos dispositivos de manera virtual para volcar allí nuestra app y hacer pruebas en ellos.
* Herramientas de la plataforma En el subdirectorio Android/Sdk/platform-tools alberga una serie de herramientas útiles para la compilación y el despliegue de la app. Entre ellas, destaca adb, acrónimo de android device bridge, que se para comunicar nuestro ordenador y nuestro móvil, permitiendo la transferencia de ficheros y la ejecución de instrucciones en el dispositivo desde nuestro terminal del PC.

Cuando creamos una aplicación en Android, lo hacemos siguiendo el asistente del IDE, el cual nos solicita algunos atributos claves y genera un montón de ficheros y directorios que constituyen la base o el esqueleto de nuestra aplicación. Vamos a desmigar un poco los atributos de interés y la organización inherente a cualquier proyecto Android dado:

* minSkdVersion: Indica el API a partir del cual nuestro programa estará disponible. Es decir, si ponemos la 15, móviles con la versión 14 o inferiores no podrán instalar o siquiera ver en la tienda nuestra aplicación. Para decidir este atributo, es bueno basarse en las estadísticas que proporciona Android sobre el porcentaje de cada versión en uso.

En teoría es deseable abarcar el máximo número de dispositivos posibles, con lo que está bien guiarse por estos datos para fijar el valor del minSdkVersion. A veces, cuando queramos emplear una característica moderna, sólo disponible en SDK tardíos, nos veremos obligados a modificar este atributo o a gestionar desde el código distintos caminos en función de la versión del dispositivo sobre la que corre nuestro aplicativo.

* compileSdkVersion. Versión del SDK con la que se está compilando nuestra aplicación. Normalmente, es la última disponible, ya que, por el principio de compatibilidad hacia atrás, una aplicación construida con la versión más moderna del API garantiza en principio que pueda ejecutarse en los dispositivos más antiguos.
* targetSdkVersion. Versión del SDK para la que se asegura compatibilidad por parte del desarrollador, por haber sido empleada durante toda la fase de pruebas. En general, su valor es igual al anterior. El sistema operativo puede usar este atributo a modo de consulta, para gestionar algunas características modernas sólo aplicables a determinado API (por ejemplo, temas o aspectos de apariencia). En caso de no definirse, se asume el valor del minSdkVersion.
* versionCode. Número entero positivo que identifica la versión de nuestra app. Seguirá la secuencia 1, 2, 3, etc. y por cada nueva evolución que queramos publicar, deberemos incrementarla.
* versionName. Empieza valiendo "1.0" por defecto y es el nombre de la versión de nuestra app. Al ser un literal, aquí podemos jugar un poco bautizar a nuestra obra con algún nombre algo más descriptivo, al estilo de los nombres dulces del SDK. Aunque para proyectos formales, se invita a seguir la nomenclatura Numero.Numero.Numero, por ejemplo 2.3.1, donde los números más a la izquierda indiquen cambios notables y los más a la derecha evoluciones más leves o correcciones insignificantes. Ambos atributos serán visibles en la descripción de las aplicaciones instaladas en el dispositivo, desde el menú Ajustes.
* applicationId. El identificador de nuestra aplicación en la tienda de Google. Debe ser, por tanto, único y distinto para cada aplicación publicada. Se le llama también paquete, por estar conformado al estilo de la nomenclatura de los paquetes Java <carpeta>. <subcarpeta1>.<subcarpeta2>.\* y porque, además, definirá cuál es el paquete raíz del código fuente en mi aplicación debajo de la carpeta src.

Como detalle sin aparente importancia, se suele dejar el que proporciona el asistente del tipo com.example.<nombre\_de\_mi\_app> lo cual resulta un revés a la hora de publicar nuestra aplicación en la tienda, ya que no se permiten identificadores del estilo com.example.\* Conviene, por tanto, echarle una pensada al inicio o tenerlo presente. Algunos ejemplos con aplicaciones:

* com.val.ebtm donde com, es abreviatura de comercial, val, derivado mi nombre, ebtm, acrónimo del nombre de mi aplicación "El boxeo tiene música".
* edu.val.idel.rivas donde edu, es el prefijo por ser un parte de proyecto educativo, val, de nuevo por mis iniciales, idel, por la compañía contratista, y rivas, del municipio madrileño sobre el que versa.

El AndroidManifest.xml es fichero clave en la configuración del entorno Android. Su ubicación es invariante (/app/src/main/) y en él se declaran los aspectos principales de una aplicación. La extensión de este archivo, dan una idea de cómo de grande o compleja es una app. Para su edición, resulta muy cómodo el asistente que incorpora Android Studio, pues como una especie de diccionario, nos predice las opciones a redactar según el contexto/posición del puntero de escritura. Algunos elementos de relevancia que deben declararse en el fichero de manifiesto:

* Permisos: algunas acciones llevadas a cabo por nuestra aplicación deben de ser consentidas y/o comunicadas al usuario. Por ejemplo, si nuestra aplicación necesita conectarse a Internet, leer la agenda de contactos o acceder a la galería, ello implicará que se describan esos permisos de forma explícita en la sección superior del manifest.
* Componentes: en Android, existe una categoría llamada componentes, que agrupa a tres tipos de clases fundamentales: las actividades, los servicios y los receptores. Las actividades, serán las distintas pantallas o vistas que compongan nuestra aplicación. Los servicios, serán clases que desarrollan algún proceso de interés de modo no visual. Y los receptores, serán una especie de sensores, que serán activados ante una alarma, el inicio del dispositivo móvil, la finalización de la descarga de un archivo y eventos del estilo.

Bien estos tres tipos de clases (Activity, Service y Receiver), que reciben el nombre genérico de componente, constituyen la parte esencial del núcleo de Android. Dominarlas, será dominar Android. Pues bien, dada su especialidad, estas deban ser declaradas de forma explícita en el fichero de manifiesto, entre otras cosas porque será Android el que lea este fichero e instancie las clases en él descritas. Si estás familiarizado con ello, esto no es ni más ni menos que la Inversión de Control o IOC.

Otros aspectos que pueden definirse en este fichero, será la configuración vertical y horizontal de una pantalla, su estilo, el icono de la aplicación, claves para el uso de APIs de Google y un sinfín de características relacionadas con la configuración.

# **3. Estructura de un proyecto Android**

Una de las características comunes de los marcos de desarrollo es que definen una estructura de directorios común a todas las aplicaciones. Así se evita que cada programador de Android establezca de forma arbitraria la ubicación de su código fuente, los archivos de recursos, etc. Además, que, cuando Android arranque, irá a buscar cada tipo de archivo a una ubicación concreta. Debemos, por tanto, conocer la convención a este respecto. A continuación, describimos las principales carpetas y ubicaciones de un proyecto Android:

* /app/build.gradle .- Fichero de configuración gradle de mi app (gestión de dependencias, versiones)
* /app/build/ .- Directorio para los .class
* /app/libs/ .- Directorio para importar librerías manualmente o privadas
* /app/src/androidTest/ .- Directorio para clases de test lanzadas desde el dispositivo o instrumentales
* /app/src/main/assets .- Para ficheros de propiedades o html
* /app/src/main/AndroidManifest.xml .- El fichero de manifiesto
* /app/src/main/java .- Directorio para el código fuente
* /app/src/main/test .- Directorio para clases de test lanzadas desde el PC o unitarios
* /app/src/main/res .- Recursos: imágenes, estilos, literales, etc.
  + /drawable/ .- Recursos gráficos genéricos
    - /drawable-ldpi (densidad baja)
    - /drawable-mdpi (densidad media)
    - /drawable-hdpi (densidad alta)
    - /drawable-xhdpi (densidad muy alta)
    - /drawable-xxhdpi (densidad muy muy alta)
  + /mimap/ .- Reservada para el icono de la aplicación
  + /layout/ .- Todos los xml que representan la interfaz de usuario
  + /anim/ .- Directorio para animaciones
  + /menu/ .- Directorio que contiene los menús de la aplicación
  + /values/colors.xml .- Archivo que define los colores empleados en la aplicación
  + /values/strings.xml .- Archivo con expresiones literales
  + /values/arrays.xml .- Archivos con array de valores estáticos o de carga
  + /values/dimens.xml .- Archivo para definir tamaños

Todos los valores que encontremos en estos ficheros precedidos de una arroba @, serán recursos que podrán ser referidos desde la clase R.

### **Clase R**

Todos los archivos ubicados bajo la carpeta /res y los valores en ellos definidos, tienen su equivalencia automática en un atributo de la clase R. Esta clase se recrea automáticamente cada vez que incluyo o modifico algún archivo de ese directorio.

La clase R es en realidad una facilidad que me propone el entorno, de modo que el uso y manejo de los recursos de la aplicación, sea más sencillo para el programador desde el código fuente. Los recursos engloban elementos tales como: ficheros, imágenes, sonidos, plantillas, botones, estilos, literales, etc.

Para que cada recurso se identifique por un número, Android crea un atributo de tipo entero en la clase R (con un valor único y arbitrario), pero con el nombre que yo he definido. El valor final del número, generalmente largo y extraño, no me importa, pues pese a que ese número será el id del recurso en cuestión en mi aplicación, precisamente gracias a este artificio podré acceder a él por su nombre.

A continuación, unos ejemplos de lo comentado:

#### **fichero tipo int**

res/layout/fila\_listado.xml → R.layout.fila\_listado

res/raw/sonido\_aviso.mp3 → R.raw.sonido\_aviso

res/drawable/icono\_notificacion.jpeg → R.drawable.icono\_notificacion

#### **atributo/elemento**

tipo int <Button android:id="@id/boton\_aceptar"></Button>→ R.id.boton\_aceptar

<string name="saludo">¡BIENVENIDO!</string> → R.string.saludo

<color name="color\_gris">#949494</color> → R.color.color\_gris

Es de vital importancia respetar la nomenclatura, de modo que todos los archivos definidos bajo la carpeta /res no contengan mayúsculas ni espacios. Siempre deberán emplearse nombres en minúscula concatenados opcionalmente por guiones bajos. En caso contrario, la generación de la clase R resultará fallida.

Al ser esta clase generada automáticamente por el entorno, es relativamente frecuente que topemos con fallos causados por su cálculo, del tipo "no existe la clase R" o "no se pudo encontrar tal o cual identificador". Cientos de comentarios en foros de Internet denuncian el fallo y aportan soluciones, que van desde reiniciar el IDE, hasta forzar la recopilación del proyecto. No hay una fórmula mágica y deberemos probar con las alternativas encontradas navegando en búsqueda de ayuda.

### **Depuración**

Depurar un programa es ejecutarlo paso a paso, pudiendo observar los cambios en los estados de las variables y siguiendo el flujo de la ejecución al detalle. Por supuesto, el IDE de Android Studio, como entorno moderno, incorpora funcionalidades que nos ayudan a realizar la depuración de nuestro código. Y ello es, junto con el Log, las herramientas fundamentales que tenemos a la hora de localizar fallos y subsanarlos. Por muy súper programador que sea uno, depurar el código será una tarea inevitable y consustancial al desarrollo.

### **Registro O LOG**

El registro de las acciones que se van acometiendo durante la ejecución de una aplicación es una necesidad inherente a cualquier sistema, por sencillo o sofisticado que sea. Es como la caja negra de un avión, o las migas en el cuento de Pulgarcito. Necesito un rastro, un registro, en inglés un log, que me ayude a visualizar qué acciones se ha llevado a cabo, cuáles no y si se ha producido alguna incidencia o error durante la marcha.

Pues bien, para satisfacer esa funcionalidad, los creadores de Android nos ofrecieren en el SDK la clase Log del paquete android.util. Se recomienda encarecidamente su uso, especialmente en los partes funcionalmente importantes o susceptibles de error, pues aporta legibilidad y sostenibilidad. Aspectos claves en la calidad de una aplicación.

Sólo es necesario invocar a uno de los métodos estáticos definidos, previstos según la importancia o criticidad del mensaje a registrar. Su uso es sencillo, yendo de menos a más prioridad:

*Log.v (String etiqueta, String mensaje) para mensajes con importancia VERBOSE*

*Log.d (String etiqueta, String mensaje) para mensajes con importancia DEBUG*

*Log.i (String etiqueta, String mensaje) para mensajes con importancia INFO*

*Log.e (String etiqueta, String mensaje) para mensajes con importancia ERROR*

*Log.e (String etiqueta, String mensaje, Throwable error)*

El parámetro etiqueta es un texto que acompañará al mensaje que queremos registrar y que se recomienda declarar como constante para ser usada a lo largo del código. Nos servirá para identificar los mensajes propios de nuestra app, al margen de los cientos de mensajes de log vertidos por otras aplicaciones en el dispositivo.

*public static final String MI\_ETIQUETA\_LOG = "APP\_VAL";*

El segundo parámetro, es el mensaje que queremos registrar. Un ejemplo sería: "Aplicación iniciada".

En el método e, el tercer parámetro viene a representar el fallo o error producidos en caso de excepción y describe el propio suceso anómalo.

# **4. Componentes de una aplicación Android**

### **Actividades**

Una Actividad es una clase que representa una pantalla. Se llaman Actividades puesto que heredan de Activity, una clase dada en el SDK. Cuando programemos nuestras actividades, tenemos que realizar dos fases de la forma más independientemente y secuencial posible: la fase estática primero y la fase dinámica después. El método expresado a continuación parece sencillo y hasta obvio, pero por mi experiencia, resulta clave en la consecución del éxito para programar nuestra actividad. Y él no seguirlo, deriva generalmente en fracaso. Esta forma procedimental nos ayudará a definir de manera completa qué queremos, la funcionalidad esperada y la forma de representarlo:

1. La fase estática: Realizar un diseño visual lo más detallado posible de los elementos de la pantalla. Qué botones, qué texto, qué menú e imágenes tendrá nuestra actividad; las opciones y cuál será su distribución en la pantalla. Todo ello, se traducirá en el producto final de esta fase que es archivo XML de layout. Pero es muy recomendable que tomes papel y lápiz y hagas un croquis lo más detallado posible antes de tocar el ordenador. O un dibujo con cualquier programa informático. Una vez tengas claro cuál quieres que sea la apariencia, la pasaremos al XML que Android entiende. Interesante en esta fase aportar datos ficticios (aunque aún no se disponga de ellos), para ver cómo quedan dibujados en la pantalla.
2. La fase dinámica: Dar vida al dibujo anterior, programando su funcionalidad asociada. Ello implicará programar nuestros métodos e instrucciones en la(s) clase(s) Java. Es decir, todas las acciones que se pueden realizar desde el interfaz gráfico definido en la fase anterior tendrán su traducción en el código que ejecute las acciones necesarias.

### **Concepto Clave Clase**

Al usar Java, que es un lenguaje orientado a objetos, nuestro proyecto estará compuesto por decenas de clases. Y, ¿qué es una clase, además del átomo de cualquier aplicación Android? Bien, la clase sirve para abstraer y representar esa parte del mundo real sobre la que versa nuestra app. De modo que no se te olvide nunca y tenlo siempre presente: una clase puede ser definida únicamente por dos motivos:

* Bien porque representa algo, por su naturaleza. Es decir, podemos por ejemplo tener una clase de un equipo de fútbol, la clasificación o un jugador, si hacemos un programa sobre La Liga o la clase Ingrediente o Receta si hacemos una aplicación sobre cocina.
* Bien por su funcionalidad, porque vale para algo. Es decir, una clase que sirve para llevar a cabo una acción: enviar datos por Internet, guardar datos en la memoria, grabar un mensaje de voz, etc.

### **Creación de una Actividad**

En realidad, cuando definimos una actividad, no sólo tenemos que definir una clase. Por tratarse de una clase especial para Android, se hacen tres cosas:

1. Definir una clase que herede de AppCompatActivity
2. Definir un elemento XML en el manifest
3. Definir un archivo XML llamado layout

Ya que la definición de una nueva clase implica estas tres acciones, es imperativo realizar esta acción mediante el asistente de Android Studio. Se puede hacer manualmente, pero no compensa. El elemento en el archivo de manifiesto no es ni más ni menos que una declaración para que el motor de Android sepa que tiene una pantalla y qué clase es la que tendrá que instanciar llegado el momento.

### **Métodos importantes en una actividad**

En una clase, podremos definir todos los métodos que consideremos necesario; pero al ser una clase que hereda de otra, ya tenemos métodos heredados que nos son dados. Entre ellos, destacan:

#### **onCreate (Bundle bundle)**

Se interpreta como nuestro método main, es decir, por donde comienza a ejecutarse nuestra aplicación. En realidad, ya veremos que es sólo una etapa más ciclo de vida de una Actividad. Pero podemos anticipar que, en este método, se prepara lo que más tarde será visible por el usuario.

#### **findViewByld(int id)**

Método básico que nos permite obtener un objeto visual de la pantalla en la que estamos. Lo buscaremos pasando al método el id o identificador del objeto. Este id, es un nombre que le damos al elemento en el XML y que la clase R convierte en un número automáticamente. Por ejemplo, si quiero obtener el mensaje que introdujo un usuario supongamos, para enviarlo; lo primero será obtener la caja de texto que contiene el mensaje mediante este método. Esa caja de texto estará definida de antemano y tendrá su id, que habremos definido en el XML.

#### **setContentView (int id)**

Método que siempre será invocado en el interior de onCreate(). Su cometido es cargar el archivo de layout que aparecerá dibujado en nuestra pantalla o Activity. Ese archivo de layout, será identificado por el parámetro id, también generado por la clase R, a partir de su nombre.

#### **finish()**

Este método finaliza la actividad, provocando que deje de verse la pantalla/actividad visible. Si es la última, se sale de la aplicación. Si no, tiene como un efecto de ir hacia atrás, pasandeo a ser visible la actividad anterior.

### **Views y ViewGroup**

Imagina todos los objetos visuales que puedes encontrar en una pantalla: desde un botón, hasta una imagen, pasando por un menú. Pues bien, todos ellos, caen dentro de la categoría de vista o View en inglés. Y todos estos elementos visuales serán clases (Button, ImageView, Menu), que heredan de View.

Además, hay un tipo especial de vista, que son las que permiten agrupar o agregar dentro de ella a otras vistas. Imagina, por ejemplo, tu lista de contactos de WhatsApp. Cada ítem o elemento de la lista, representa un contacto, pero todos ellos, están englobados en un elemento de entidad superior que es la propia lista. Pues bien, esa lista es un ViewGroup, que no es sino un tipo particular de Vista.

Views y ViewGroups, componen una estructura jerárquica donde puede entenderse que unos elementos <<hijos>> penden de otros <<padres>> que los agrupan y organizan. Esta relación quedará textualmente reflejada en un fichero XML de layout.

### **Atributos Comunes de una Vista**

Cada ViewGroup o View, tendrá sus propios campos exclusivos en función de su subtipo (botón, caja de texto, imagen, etc.) pero sí que, como tipos concretos de Vista, comparten una serie de atributos comunes de especial relevancia que debemos conocer:

* **android:id** Identificador. Indispensable para poder referirse a la vista desde el código y distinguirla de las demás. El nombre que demos aquí tendrá su equivalente numérico en la clase R de modo automático.
* **android:layout\_width** Ancho. Cuál es la medida de ancho de un elemento.
* **android:layout\_height** Alto. Cuál es la medida de alto de un elemento.
* **android:gravity** Posición de un elemento (centrado, izquierda, derecha, etc.) respecto de sus márgenes.
* **android:layout\_gravity** Posición de un elemento (centrado, izquierda, derecha, etc.) respecto de su padre/elemento contenedor.
* **android:margin** Margen respecto de sus elementos circundantes (marginBottom –margen respecto del elemento inferior –, marginTop –margen respecto del elemento superior –, marginLeft –margen respecto del elemento a la izquierda –, marginRight –margen respecto del elemento a la derecha –). Digamos este es el margen externo.
* **android:padding** Margen interno, respecto de sus límites. Igual que para el atributo anterior, hay opción de definir individualmente margen superior, inferior y laterales.

Al par de atributos android:layout\_width y android:layout\_height (ancho y alto respectivamente) se les denomina en conjunto como LayoutParams y son obligatorios para toda vista.

### **Unidades de Medida**

A la hora de definir el tamaño de un elemento, su margen, ancho o alto debo usar la medida dp, que se escala automáticamente en función de la resolución del dispositivo. Como curiosidad dp proviene de pixels de densidad independiente.

Sólo para texto, debo emplear sp, que es como dp (adaptada a la resolución) pero además tiene en cuanta el tamaño de fuente elegido por el usuario en Ajustes (letra grande, muy grande, pequeña, etc.).

También puedo usar dos medidas relativas súper útiles para definir el tamaño de un control: match\_parent y wrap\_content. Si atribuyo la primera de ellas a un elemento, este ocupará todo el espacio disponible. Mientras que, si uso la segunda, ocupara lo estrictamente necesario pare representarse.

Toda Actividad va a llevar asociada un archivo de diseño o layout donde están declaradas las vistas que conforman el conjunto visual de la pantalla. Este archivo puede ser modificado e incluso creado sobre la marcha (como veremos en la sección de Inflar) aunque lo normal, es partir de un diseño definido en tiempo de compilación.

### **Layout Básicos**

Un Layout (plano o diseño, del inglés) es un elemento no visual dedicado a establecer la distribución y posición de los elementos que se definen en su interior. Si recuerdas la definición del tema anterior, efectivamente, un layout es un ViewGroup y por ende un caso particular de View, ya que agrupará un conjunto de controles declarados jerárquicamente como hijos suyos.

Algunos de los layouts más relevantes son los siguientes:

* LinearLayout: Los elementos en él definidos aparecen uno a continuación del otro. Bien en horizontal, bien en vertical, según el valor del atributo orientation. Es un clásico y combinándolo de forma anidada se puede hacer virguerías.
* ScrollView: Usado para hacer que cuando el contenido de una actividad sobrepasa el tamaño de la pantalla, ésta deslice. Normalmente agrupa a otro layout y sólo puede tener un hijo directo como restricción. El desbordamiento puede ser horizontal o vertical.
* FrameLayout: Sirve para superponer elementos. Los controles definidos en su interior aparecen dibujados uno sobre el otro, según el orden de aparición. Se puede usar para crear fondos, transparencias, etc.
* RelativeLayout: Cuando sitúo un elemento, lo estaré haciendo respecto de su hermano o padre. La posición estará en función de otro elemento definido previamente (a su izquierda, a su derecha, a continuación). De ahí lo de relative.
* ConstraintLayout: Parecido a la anterior, es la más moderna de las distribuciones. Está pensada para que pueda construirse la pantalla de modo gráfico, facilitando así el modelado a programadores noveles. Coloco los elementos arrastrándolos desde el menú. Es imprescindible definir al menos una restricción vertical y otra horizontal para cada elemento. Restricción toma aquí el sentido de distancia obligatoria a un borde de la pantalla u otro elemento.

Hay otras distribuciones que están demodé (TableLayout, GridLaoyut, ListView) al haber sido sustituidas por RecyclerView. Dicho layout, se emplea para mostrar colecciones de datos, como una lista de productos, contactos o cualquier otro tipo que se te ocurra. Todas estas distribuciones o ViewGroups, albergarán en su interior controles o Views más sencillas.

### **Callback y Listener**

Hay dos conceptos tradicionales en la programación que toman vital relevancia en Android. La llamada por detrás o callback y el escuchador o listener. Íntimamente relacionados y complementarios.

Imagina que tocas un botón y se reproduce una canción, que tocas una flecha y se envía un mensaje, o que deslizas un elemento de una lista y desaparece, por cada una de esas potenciales acciones, hay un proceso al acecho, que está esperando a que se produzcan para poder desencadenar las acciones asociadas. Ese proceso que espera, que está a la escucha de los gestos sobre un botón, imagen o cualquier otro elemento visual, es lo que entendemos como listener.

Cuando se dispara uno de estos eventos ejecutándose el método establecido. Esa llamada a nuestro método callback. Se llama así ya que generalmente nuestra función será invocada desde Android.

### **Visibilidad: Visible, Invisible y Gone**

Un objeto visual en la pantalla puede transitar entre estos tres estados de visibilidad. Por defecto, todo elemento visual, al declararse en el fichero XML de layout, está en estado visible. Podemos hacer que desaparezca temporalmente (pasándolo a invisible) y podemos eliminarlo de la jerarquía, borrarlo del todo (pasándolo a gone). Es una necesidad habitual en las aplicaciones.

### **Ciclo de Vida de una Actividad**

Una actividad evoluciona desde que va a ser dibujada en la pantalla hasta que deja de ser visible. En ese tránsito, pasa por un conjunto de estados bien definidos, que se corresponden con la ejecución de una serie de métodos. Estos métodos son invocados de modo secuencial y automático por Android.

Cada método aporta una funcionalidad implementada ya en la clase padre. En caso de que necesitemos modificar o ampliar el comportamiento de estos métodos, podemos sobrescribirlos en nuestra actividad, añadiendo en ellos el código que queramos.

El usuario entra en nuestra actividad, rellena parcialmente el formulario y abandona. Por defecto, los datos introducidos se perderán, ya que, al salir, el método que se ejecuta de modo automático es onStop() y este no preserva la información introducida. Seremos nosotros como programadores, los que sobrescribiendo el método onStop() en nuestra clase, podemos añadir las instrucciones con las almacenar los datos que deseamos preservar.

### Estados de una Actividad

* **onCreate** Inicializa. Prepara lo que se va a ver. A modo de constructor (sin serlo, ojo)
* **onStart** La actividad se hace visible
* **onResume** La actividad está en primer plano
* **onPause** La actividad está apilada / parcialmente visible
* **onStop** Actividad no visible
* **onDestroy** Actividad finalizada