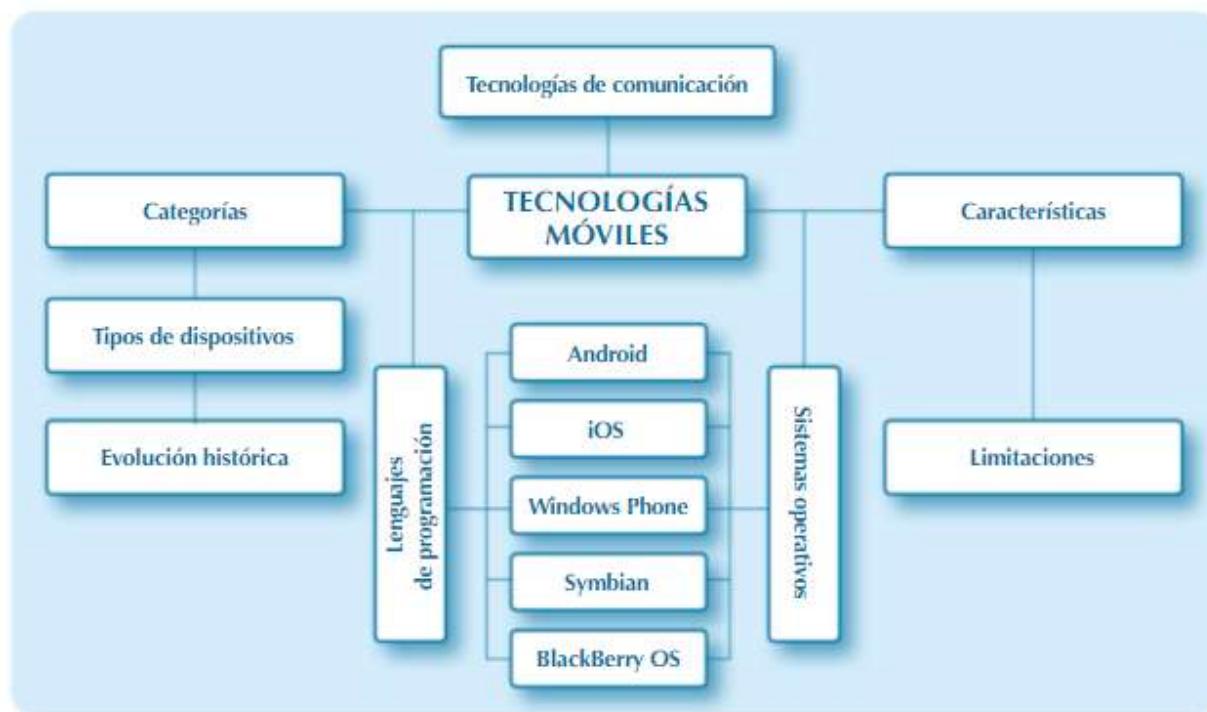


Mapa conceptual



Glosario

API (Application Programming Interface). La interfaz de programación de aplicaciones es un conjunto de funciones y procedimientos cuyo uso permite a los desarrolladores interactuar con determinados elementos del sistema operativo o de otros programas.

APP. Acrónimo de aplicación informática.

Framework. Entorno o marco de trabajo formado por un conjunto de elementos o módulos de software que sirven como base para la organización y desarrollo de software.

Gadget. Dispositivo, generalmente tecnológico y novedoso, de pequeñas proporciones que tiene función específica y práctica.

Kernel. Núcleo, parte central y fundamental del sistema operativo. Suele ser responsable de la comunicación entre el software y el hardware del sistema informático.

Runtime. Intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.

SMS (Short Message Service). Servicio de mensajes, disponible en los teléfonos móviles, que permite el envío de mensajes cortos entre dispositivos móviles.

UI (User Interface). Interfaz del usuario. Es la vista que permite al usuario interactuar de manera efectiva y cómoda con un sistema.

WAP (Wireless Application Protocol). Protocolo de aplicaciones inalámbricas. Es un estándar abierto internacional para aplicaciones y dispositivos que utilizan comunicaciones inalámbricas.

1.1. Introducción

Aunque los rápidos cambios a los que nos tienen acostumbrados las tecnologías, especialmente las de la información, han hecho que sepamos adaptar, a vertiginoso ritmo, nuestros hábitos y rutinas de vida, quizás ninguno de ellos ha producido una variación tan radical en nuestra mentalidad y conducta como la consecuente a la gran revolución que han supuesto las redes móviles y, especialmente, la conjugación con ellas de teléfonos inteligentes, capaces de simplificar en un solo dispositivo multitud de tareas para las que hasta ahora necesitábamos variados y complejos aparatos.

1.2. Tecnologías móviles: características y limitaciones

Antes de adentrarnos en el desarrollo de aplicaciones móviles, es conveniente conocer las tecnologías móviles que pueden encontrarse en el mercado, dispositivos que las soportan y aquellos otros aspectos que pueden ser relevantes a la hora de programar una aplicación móvil.

1.2.1. Características de los dispositivos móviles

Una primera aproximación al concepto de *dispositivo móvil* nos podría hacer definirlo como aquel aparato con capacidad de procesado y almacenamiento, de pequeño tamaño, portable, autonomía de funcionamiento en cuanto a batería, conexión permanente o semipermanente a una red de comunicaciones y con variadas funciones, si bien una de ellas es para la cual fue fundamentalmente diseñado. Esta definición nos muestra ya alguna de sus características generales, que podríamos agrupar en cuatro grandes bloques.

1. *Capacidad de procesado*: puesto que, en la actualidad, relacionamos el concepto de dispositivo móvil, en cualquiera de sus variantes, con el de microordenador, con cierta especificidad de funciones. Esto lo asocia a capacidad de cálculo (en forma de velocidad de procesado) y almacenamiento, que incluiría el necesario para el procesado y la persistencia de la información en él depositada (no menos importante).
2. *Tamaño*: el concepto de portabilidad lo llevamos a sus máximos extremos, alejándonos de los ordenadores portátiles, en cuanto les exigimos que estos puedan compartir con el usuario su vida diaria, independientemente de la actividad que se esté realizando, aunque esta sea deportiva.
3. *Movilidad*: asociada a la anterior característica, pero también a la no dependencia de cableado para su alimentación o comunicación. Es decir, pequeño tamaño, batería duradera y comunicación inalámbrica son los tres pilares sobre los que se asienta esta propiedad.
4. *Conectividad*: si bien no es novedosa la comunicación inalámbrica entre teléfonos, utilizando variadas tecnologías (muchas de ellas ya obsoletas), han sido los avances en las redes móviles y la masiva incorporación del uso de redes de datos lo que ha creado un nuevo paradigma a la hora de definir estos dispositivos.

Además de estas propiedades, podríamos incluir muchas otras, como sistema operativo, diseño, ergonomía, tamaño de pantalla, gadgets incorporados..., todos ellos importantes y posiblemente, en algunos casos, decisivos a la hora de adquirir uno de estos aparatos, pero consideradas como propiedades secundarias a la hora de su clasificación.

1.2.2. Tipos de dispositivos móviles

Basándonos en las anteriores características, podemos agruparlos en tres categorías, según los estándares que en 2005 propusieron T38 y DuPont Global Mobility Innovation Team:

- *Limited Data Mobile Device (Dispositivo Móvil de Datos Limitados)*: dispositivos de pantalla pequeña, generalmente tipo texto, y servicios de datos limitados a SMS y acceso WAP.
- *Basic Data Mobile Device (Dispositivo Móvil de Datos Básicos)*: pantalla de mediano tamaño (entre 120×120 y 240×240 píxeles), con menú de navegación basado en interface gráfica, posibilidad de correo electrónico y navegación web.
- *Enhanced Data Mobile Device (Dispositivo Móvil de Datos Mejorados)*: pantallas de medianas a grandes, con las mismas características que los anteriores, a las que se añaden variadas aplicaciones nativas, corporativas y multitud de gadgets, como cámara, GPS, sensores.

En cuanto a los tipos, cuando utilizamos el término *dispositivo móvil* nos referimos a grupo amplio de aparatos electrónicos. Teniendo en cuenta lo anteriormente visto y desde un punto de vista de clasificación, podemos agrupar los dispositivos móviles en los siguientes tipos:

- a) *Teléfonos móviles*: incluimos en este apartado aquellos teléfonos cuya funcionalidad se limita básicamente a la comunicación por voz, si bien en algunos casos pueden incluir operatividad SMS, cámara fotográfica, agenda.
- b) *Handhelds*: con variado diseño, inicialmente con pantalla plegable sobre el teclado, fueron utilizados como agenda/organizador con posibilidad de ejecutar algunas aplicaciones (científicas o financieras). Lanzados por Psion, Casio, tenían sus propios sistemas operativos de baja compatibilidad, hasta que Microsoft lanzó Windows CE, que fue adoptado por la mayoría de las compañías. Durante un tiempo, fueron utilizados como complemento de los ordenadores portátiles, pero a pesar de la evolución de los sistemas operativos (Windows Mobile, CE 202), la mayoría de las compañías fueron abandonando este formato de dispositivos.
- c) *Netbooks*: lanzados al mercado en 1999 por Psion y con buena acogida hasta 2007, se diseñaron originariamente como ordenadores compactos de pequeño tamaño y bajo precio con suficiente poder de cálculo como para darle uso ofimático y tener acceso a Internet.



PARA SABER MÁS

El término *nettop* se aplica a aquellos equipos de características similares en cuanto al tamaño y las prestaciones a los netbooks. No son portátiles, sino de escritorio.

- d) *E-Book Readers*: con variadas denominaciones (libro electrónico, eBook, libro digital), son dispositivos usados para leer ficheros de documentos (generalmente libros) electrónicos. Suelen ser ligeros, con alta capacidad de almacenaje, bajo consumo y normalmente utilizan una tecnología de pantalla denominada tinta electrónica.

- e) *Tablets*: con tamaño intermedio entre el ordenador y el móvil, tienen como principales características la ligereza, autonomía y fácil manejo (pantalla táctil). Aunque cuentan con algún antecedente histórico, se considera que su irrupción en el mercado data de principios de este siglo, con los lanzamientos de Microsoft Tablet PC, la Nokia 510 webtablet o el iPad de Apple.

Variantes de estas son las minitablets, que tienen un tamaño de 7 u 8 pulgadas, los “tabléfonos”, popularmente denominados a los smartphones de gran tamaño. El portátil convertible cuyo teclado se desliza (en algunos incluso puede separarse) debajo de la pantalla, que suele ser táctil, pudiéndose utilizar como una tableta. Los booklets, dispositivos con dos pantallas, una de las cuales es táctil, que se comporta como teclado virtual.



Figura 1.1
Tipos de dispositivos móviles.

- f) *Personal digital assistant*: más conocidos como PDA, son dispositivos que intentan combinar variadas prestaciones con la pretensión de funcionar como un organizador digital. Inicialmente manejados con un lápiz (stylus), fueron evolucionando en la calidad y resolución de pantalla. Los más conocidos fueron los Palm y el Pocket PC, sin embargo, tanto uno como otro decayeron con la eclosión en el mercado de los teléfonos móviles inteligentes.
- g) *Smartphones*: dispositivos cuya principal funcionalidad es la de teléfono móvil, pero que por sus características, muy similares a las de un ordenador personal, y la variedad de recursos electrónicos que progresivamente se le han ido añadiendo, han hecho que la mayoría de los usuarios recurran a ellos para obtener prestaciones, en muchos casos distintas a las de comunicación telefónica. Suelen ser características comunes a todos ellos las pantallas de alta calidad, la múltiple conectividad, la gran capacidad de procesamiento y almacenaje, y la riqueza en dispositivos incorporados (cámara, sensores, GPS).

Pueden encontrarse en múltiples sistemas operativos y versiones de los mismos, pero en todos los casos tienen la posibilidad de incorporar nuevas aplicaciones que generalmente se suelen descargar de repositorios públicos.

- h) *Gadgets*: dispositivos electrónicos con una función específica, generalmente pequeños, prácticos y novedosos. Tradicionalmente se han englobado en ellos algunos de los ya vistos, como PDA, móviles, smartphones, reproductores mp3, pero cada vez es más frecuente el utilizar este término para dispositivos que aplican las últimas tecnologías a la electrónica de consumo.

Actividad propuesta 1.1



Localiza a través de Internet cinco E-Books y realiza una tabla donde se muestren las siguientes características: fabricante, tipo de pantalla, tamaño de la pantalla, conectividad, memoria del almacenamiento, autonomía.

1.2.3. Tecnología de comunicación móvil

La red de comunicación móvil ha ido evolucionando con el paso del tiempo. Cada generación aporta cambios a la anterior:

1. *Generación 0.* Aunque existe cierta discrepancia sobre su consideración como dispositivos móviles, se incluyen aquí aquellos dispositivos que utilizaban ondas de radio para comunicarse, como los Walkie Talkie. Los estándares de comunicación en esta generación son:
 - PTT: siglas de Push to Talk, *pulsar para hablar*.
 - IMTS: Improved Mobile Telephone System.
2. *Primera generación (tecnología 1G).* Surge a partir de 1977. Se trata de equipos analógicos de gran tamaño y peso, mecanismo de transmisión y recepción con ondas de radio, que solo podían ser utilizados para la transmisión de voz y tenían baja seguridad. La primera red celular se implantó en el año 1977 en Chicago, siendo Japón el primer país con red 1G nacional. Fue la primera generación considerada realmente como de teléfonos móviles, que contaba con los siguientes estándares:
 - AMPS (American Advanced Mobile Phone Service): en América del Norte. Introducido en 1987.
 - NMT (Nordic Mobile Telephone): para los países nórdicos. Introducido en 1979.
 - TACS (British Total Access Communication Systems): en Inglaterra. Introducido en 1983.
 - NAMTS (Nippon Advanced Mobile Telephone Systems): en Japón. Introducido en el 1981.
3. *Segunda generación (tecnología 2G).* Tuvo su inicio sobre el año 1990, pasando así de la tecnología analógica a la digital. Surge ante la necesidad de transmitir datos y voz. Se mejora el proceso de llamadas y se integran otros servicios adicionales a la voz, de entre los que destaca el Servicio de Mensajes Cortos (Short Message Service). Una de las características principales del estándar GSM es el Módulo de Identidad del Suscriptor, conocido comúnmente como tarjeta SIM, tarjeta que contiene la información del usuario, datos de red y directorio telefónico. Las tecnologías predominantes en esta generación son:
 - GSM (Global System for Mobile Communications): utilizado en Europa.
 - IS-136 (conocido también como TIA/EIA136 o ANSI136): utilizado en EE.UU.

- CDMA (Code Division Multiple Access).
 - PDC (Personal Digital Communications): utilizado en Japón.
4. *Segunda generación y media (tecnología 2.5G-2.75G)*. Se crea ante la necesidad de incrementar la velocidad del tráfico de datos, facilitando la navegación por Internet. Pueden incluirse aquí algunos teléfonos móviles que incorporan algunas de las mejoras y tecnologías del estándar 3G. Los estándares más utilizados son:
- GPRS (General Packet Radio Service): mejora la velocidad de transmisión de datos.
 - EDGE (Enhanced Data for Global Evolution): también conocida como EGPRS.
5. *Tercera generación (tecnología 3G)*. Tiene sus orígenes en octubre del 2001 en Japón. Supone un salto cualitativo en el mundo de las telecomunicaciones, al poder, a través de un dispositivo móvil, navegar con comodidad por Internet. Esta generación está basado en los UMTS (Universal Mobile Telecommunications System). Caracterizada por la convergencia de la voz y alta transmisión de datos con acceso. Este sistema alcanzaba velocidad de transmisión de hasta 2Mbps, por lo que se usó el esquema de Acceso Múltiple por División en Códigos (CDMA).
6. *Tercera generación y media (tecnología 3.5G-3.75 G)*. Una modificación de la 3G, cambiando la tecnología UMTS a la llamada HSPA (High-Speed Downlink Packet Access), que permite que su velocidad de transmisión llegue a los 14Mbps. Esta generación también se conoce como 3G+.

Figura 1.2
Comparación entre las diferentes generaciones de redes móviles.

1G	2G	3G	4G	5G	6G
1980 Solo voz Analógico	1990 Voz y SMS Digital	2001 Voz y datos Multimedia	2010 Datos Protocolo IP	2020 Datos Banda ancha	¿? Datos Satelital
					
2,4 Kb/s	64 Kb/s	2 Mb/s	100 Mb/s	1 Gb/s	¿?

7. *Cuarta generación (tecnología 4G)*. Empezó a desplegarse a partir del año 2010. Ofrece un mayor ancho de banda que permite la recepción de televisión en alta definición. Long Term Evolution (LTE) es reconocida como 4G y mejora los sistemas basados en UMTS (Universal Mobile Telecommunications System), además, utiliza Wimax (Worldwide Interoperability for Microwave Access), que permite velocidades de descarga de hasta 60 Mbps y envíos de paquetes de información de hasta 40 Mbps.
8. *Cuarta generación y media (tecnología 4G+)*. Aparecida el 2016. Supone un avance en la velocidad de transmisión denominada 4G+ que puede superar los 400 Mbps.
9. *Quinta generación (tecnología 5G)*. Presentada durante el Mobile World Congress celebrado en Barcelona en febrero del 2017, inicia su lanzamiento comercial en 2019, fijando para 2020 su despliegue generalizado. Esta es una tecnología que multiplica por diez la velocidad del 4G (llegando a 1 Gbps de velocidad de transmisión). La quinta generación, conocida también como Real Wireless World System, integra diferentes sistemas

inalámbricos, como Wi-Fi, redes celulares, sistemas de corto alcance con redes de sensores inalámbricos (WSN), comunicaciones de máquina a máquina (M2M), y genera el llamado *Internet de las Cosas* (IoT).

10. *Sexta generación (tecnología 6G)*. La sexta generación implementará los sistemas satelitales con la quinta generación, con lo cual se puede conseguir una cobertura global. Un sistema basado en redes satelitales de navegación de posicionamiento global, utilizadas en la telecomunicación para la telefonía.

Actividades propuestas



- 1.2.** Entra en la siguiente dirección web y localiza la cobertura 5G en tu región y quiénes son los proveedores que la dan: <https://www.speedtest.net/ookla-5g-map>
- 1.3.** Realiza una búsqueda a través de Internet de la evolución de las ventas de teléfonos móviles de Apple, Samsung y Huawei durante los tres últimos años.



RECURSO DIGITAL 1.1

Para saber más, véase el recurso digital *Evolución de los dispositivos móviles*.

1.2.4. Limitaciones de los dispositivos móviles

La programación de aplicaciones para móviles tiene muchos aspectos similares al desarrollo de aplicaciones para escritorio, pero es necesario entender que para conseguir que los programas funcionen correctamente en un amplio y variado número de dispositivos se debe conocer cuáles son las limitaciones o restricciones a las que estamos sujetos.

La mayoría de las limitaciones estarán relacionadas con las características hardware. Aunque el mercado ha evolucionado de manera importante, se han de tener en cuenta las variaciones que se pueden encontrar en la capacidad de procesado y almacenamiento, evitando, en la medida que sea posible, la excesiva carga en elementos multimedia.

Debemos apartarnos de la tendencia a programar para dispositivos de alta gama, ya que si se logra que una aplicación funcione bien en teléfonos con pocos recursos, seguramente lo hará aún mejor en otros de gama superior. Siendo aconsejable, en cualquier caso, optimizar el uso del procesador para una rápida ejecución de nuestro código y respuesta de la aplicación.

La diversidad de tamaño de pantalla también puede hacernos modificar más de una vez una aplicación, ya que diferentes resoluciones harán obtener resultados muy lejanos a los deseados por el programador. Además, no todos los equipos están dotados de sensores y las tecnologías que son consideradas básicas, lo que puede provocar que ni siquiera admitan la instalación de la aplicación. Es importante conocer las diferencias entre plataformas operativas y las versiones que existan de ellas, ya que nunca se tendrá garantizada la retrocompatibilidad u operatividad futura del software desarrollado. También hay que tener cuidado al incorporar herramientas y librerías de terceras partes, puesto que no son admitidas por todos los equipos.

TOMA NOTA



Nunca un procesador más potente o una mayor existencia de memoria será la solución a una aplicación mal diseñada.

Por último, a pesar de la generalización de las redes móviles y con ella la conectividad de casi la totalidad de los dispositivos, las posibles limitaciones del ancho de banda y las desconexiones temporales de la red por falta de cobertura pueden llevar al fracaso de una aplicación si su lógica demanda una conexión constante a la red.

Por todo lo anterior, es recomendable que antes de llevar una aplicación a la fase de explotación, se pruebe y verifique en el más variado número de dispositivos posibles, utilizando, en caso de no poder hacerlo con equipos reales, los emuladores que suelen suministrarte los entornos de desarrollo.



Actividad propuesta 1.4

BQ Aquaris, Energy Phone, Wolder WIAM son algunos de los productos nacionales en el mundo de los smartphone. Localiza qué empresa nacional sigue comercializando dispositivos de marca propia, qué modelos ofertan y cuáles son sus características.

1.3. Sistemas operativos móviles

Al igual que en los ordenadores personales, el sistema operativo móvil es el conjunto de programas de bajo nivel que permiten la abstracción de las características propias del hardware y provee servicios a las aplicaciones que se ejecutan sobre él. La mayoría de los sistemas operativos móviles están basados en el modelo de capas con variada complejidad y multiplicidad a medida que se alejan de los elementos hardware. Simplificando cada una de ellas, podemos encontrar las siguientes:

- *Núcleo o kernel*. Proporciona el acceso a los elementos del hardware, ofreciendo servicios a las capas superiores a través de los controladores o drivers, la gestión de procesos, el sistema de archivos y la gestión de la memoria.
- *Middleware*. Conjunto de módulos que posibilitan la existencia de aplicaciones. Es transparente para el usuario y ofrece servicios como el motor de mensajería y comunicaciones, códec multimedia, intérpretes web, gestión del dispositivo y seguridad.
- *Entorno de ejecución de aplicaciones*. Gestor de aplicaciones e interfaces abierto que permite la programación por parte de los desarrolladores para la creación de software.
- *Interfaces de usuario*. Facilitan la relación con el usuario y se encargan de la presentación visual de la aplicación. Incluyen los componentes gráficos (botones, pantallas, listas, etc.) y el marco de interacción.
- *Aplicaciones nativas*. Propias de cada uno de los modelos y fabricantes.

1.3.1. Android

Lanzado en 2007, es el sistema operativo líder en el mercado móvil. Fue desarrollado por la Open Handset Alliance (OHA), una agrupación de 78 compañías lideradas por Google. Su principal cualidad es su carácter abierto, ya que se distribuye bajo dos tipos de licencias, una que abarca el código del kernel, la GNU GPLv2; y otra licencia para el resto de componentes del sistema, que se licencia bajo APACHE v2. La arquitectura de Android tiene las siguientes capas:

- a) *Kernel de Linux.* El tiempo de ejecución de Android (ART) se basa en el kernel de Linux, con funcionalidades como la generación de subprocessos y la administración de memoria de bajo nivel.



Investiga

Consulta las diferencias que existen entre ART (Android Runtime) y Dalvik, que fue la máquina virtual utilizada originalmente por Android.

- b) *Capa de abstracción de hardware (HAL).* Proporciona interfaces que enlazan las capacidades de hardware del dispositivo al framework de nivel más alto. Posee varias bibliotecas, con una interfaz para cada tipo de componente (cámara, bluetooth...).
- c) *Runtime.* Cada APP ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de ART, ya que está diseñado para ejecutar varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes específico de Android, y optimizado para ocupar un espacio de memoria mínimo.
- d) *Bibliotecas C/C++ nativas.* Muchos componentes y servicios se basan en código nativo, que requiere bibliotecas nativas escritas en C y C++. La plataforma Android proporciona la API para posibilitar la funcionalidad de estas bibliotecas.
- e) *Framework de la Java API.* Todo el conjunto de funciones del SO Android está disponible mediante API escritas en el lenguaje Java.
- f) *Apps del sistema.* Android incluye un conjunto de apps nativas para correo electrónico, mensajería SMS, calendarios, navegación en Internet.

Actividad propuesta 1.5



Localiza en qué versión de Android se dio soporte nativo a los sensores y en cuáles otras se incorporan Material Design, Digital Wellbeing o el Pixel Themes.

1.3.2. iOS

Desarrollado por Apple y originariamente denominado iPhone OS, es un derivado de Mac OS X. Su principal fuerte es la combinación, casi perfecta, entre hardware y software, así como el manejo de la pantalla multitáctil que lo convierte en un sistema operativo basado en la manipulación directa. Su arquitectura está basada en capas: las capas más altas contienen los servicios y tecnologías más importantes para el desarrollo de aplicaciones y las capas más bajas controlan los servicios básicos.

- *Cocoa Touch*: capa superior y más importante para el desarrollo de aplicaciones iOS. Es la que los usuarios utilizan para interactuar con las aplicaciones, es decir, la capa visible. Esta capa está formada fundamentalmente por dos Frameworks: UIKit (clases para el desarrollo de una interfaz de usuario) y Foundation Framework (acceso y manejo de objetos, servicios del SO).
- *Media Services*: provee los servicios de audio, gráficos y multimedia a la capa superior.
- *Core Services*: proporciona los servicios imprescindibles del sistema para poder ser utilizados por todas las aplicaciones (base de datos, acceso a la red).
- *Core OS*: núcleo del sistema con las características de bajo nivel (manejo de memoria, seguridad, drivers del dispositivo).

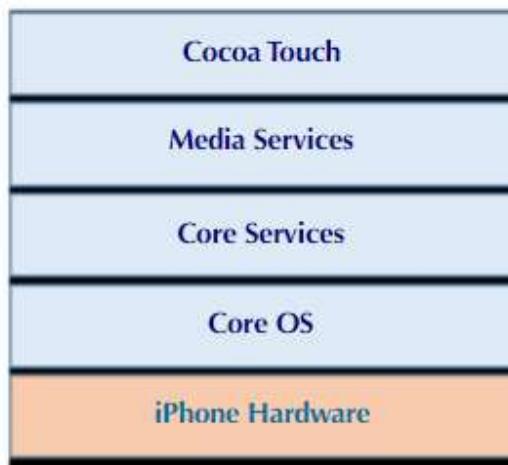


Figura 1.3
Arquitectura iOS.

Recurso web

En este código QR encontrarás un vídeo de Xataka TV con información sobre Fuchsia OS, un sistema operativo alternativo a Android en el que Google lleva trabajando desde 2016, basado en el kernel Zircon (anteriormente, Magenta):



1.3.3. Windows Phone

Desarrollado por Microsoft (antes llamado Windows Mobile), fue lanzado en el año 2010. Es un SO móvil compacto en el núcleo del sistema operativo Windows CE. Con un diseño similar a las versiones de escritorio de Windows, tiene la ventaja de la gran oferta de software de terceros. Entre sus novedades, destaca la denominada interfaz de usuario *Metro* basada en

el uso de mosaicos dinámicos con información útil para el usuario. Su arquitectura se basa en tres modelos:

- *Modelo de aplicación.* Las aplicaciones se despliegan en forma de paquete XAP, un archivo donde se encuentran los ensamblados y recursos originales de la aplicación.
- *Modelo de UI.* Modelo de interface que engloba el conjunto de interacciones que realiza un usuario sobre la aplicación.
- *Integración con la nube.* Integración con servicios como Exchange, Google Mail, Hotmail, Xbox Live, Skydrive, Facebook o Bing.

1.3.4. BlackBerry OS

Desarrollado por la RIM (*Research in Motion*), destaca por su capacidad de enviar y recibir correo electrónico a través de los operadores que ofrecen este servicio. Al igual que en Symbian, se pueden crear programas para este sistema operativo, pero es necesario tener cuenta de desarrollador de RIM.

Su arquitectura está basada en microkernel, que implementa una cantidad mínima de software en el núcleo, ejecutando otros procesos en el espacio de usuario que está fuera del kernel. Esto hace que sea menos vulnerable y más fácil de verificar. Otros elementos de su arquitectura son el gestor de arranque de la CPU, los controladores y servicios para soportar los subsistemas voz, datos, los servicios de plataforma y aplicaciones.

1.3.5. Symbian

Creado por la alianza de varias empresas de telefonía móvil (Nokia, Sony Ericsson, Samsung, Siemens), fue una evolución del sistema operativo Epoc, desarrollado por Psion en sus agendas electrónicas durante los 80. Entre sus características, podemos encontrar un eficiente uso de todos los recursos de la máquina (especialmente de la batería, la memoria RAM y la ROM).

Su arquitectura está basada en un microkernel: una mínima porción del sistema tiene privilegios de kernel, el resto se ejecuta con privilegios de usuario. Cada aplicación corre en sus propios procesos y tiene acceso solo a su propio espacio de memoria. Este sistema operativo perdió protagonismo con la llegada de iOS y Android, por lo que en la actualidad ha quedado relegado a un sector minoritario.

1.3.6. Palm OS P (WebOS)

Desarrollado originalmente en 1996 por la empresa Palm Inc para PDA's y conocido también como LG webOS, open webOS y HP webOS, es un SO multitarea basado en Linux, con una interfaz gráfica apoyada en tecnologías web (HTML5, JavaScript y CSS). Las aplicaciones se basan en el marco de una interfaz de usuario y de funciones establecidas para los servicios de acceso.

1.3.7. Firefox OS

Desarrollado por Mozilla Corporation, tiene núcleo Linux y está basado en HTML5. Está pensado para dispositivos móviles de gama baja y permite que las aplicaciones HTML5 puedan comunicarse directamente con el hardware usando JavaScript y Open Web APIs.

1.3.8. Ubuntu Touch

Desarrollado por Canonical Ltd. Está basado en Linux, aunque posee una interfaz que puede ser utilizada en ordenadores de sobremesa, portátiles, netbooks, tablets.

1.3.9. Harmony OS

Novedoso sistema operativo de Huawei Technologies Co., Ltd. Distribuido y de alto rendimiento, diseñado para el interconexión en Internet de las Cosas, al ser compatible con múltiples dispositivos. Es un sistema operativo de código abierto con arquitectura basada en microkernel, centrado en servicios básicos (solo gestiona las instrucciones del chip, la comunicación entre procesos y la información de seguridad). Un motor de compilación, ARK, establece las prioridades de ejecución de tareas, anticipándose en la reserva de los límites de tiempo. El resto de los procesos se ubican fuera del núcleo, no interfiriendo entre sí cuando están en ejecución. Pueden, además, cada uno de ellos, depurarse, desarrollarse y actualizarse por separado.



Actividad propuesta 1.6

Realiza un estudio de los sistemas operativos de móviles que más frecuentemente se venden en España, en la actualidad, y compáralo con otros países, como Japón y EE.UU.

1.4. Lenguajes de programación para dispositivos móviles

El mundo de desarrollo de aplicaciones para móviles dispone de infinidad de herramientas, lenguajes y entornos para elegir antes de sentarte a codificar. La primera opción que se ha de tomar es el tipo de lenguaje de programación por el que se va a optar.

1.4.1. Desarrollo nativo

Aunque existen defensores de cada opción, el desarrollo nativo suele ser la mejor decisión, a pesar de que cada plataforma (iOS, Android...) utiliza lenguajes y paradigmas de programación diferentes y herramientas propias, pero a cambio se obtiene gran flexibilidad, adaptación total al entorno y un máximo rendimiento.

La principal desventaja de hacer que una aplicación sea compatible con más de un entorno es que se tendrá que dominar más de un lenguaje y distintas herramientas, lo que aumentará el tiempo de desarrollo. Por ello, generalmente, los programadores se suelen especializar en una plataforma, sacándole el máximo rendimiento a la programación en la misma.

1.4.2. Desarrollo multiplataforma compilado a nativo

Una opción intermedia es utilizar plataformas mixtas que permitan independizar el desarrollo del lenguaje nativo de cada plataforma móvil. Con un único lenguaje, se podrán crear

aplicaciones para todos los sistemas operativos, aunque siempre habrá que realizar pequeñas adaptaciones.

La más conocida es Xamarin, basada en el lenguaje C# de Microsoft y en la plataforma .NET, que gracias a sus herramientas permite crear aplicaciones multiplataforma, reutilizando gran parte del código (a excepción de la interfaz).

Las aplicaciones escritas con Xamarin se compilan a código nativo en cada plataforma, lo que le da el mismo rendimiento que el de una aplicación nativa. También tiene acceso directo nativo a todas las API de cada plataforma, así como a los controles de interfaz de usuario nativos.

1.4.3. Desarrollo multiplataforma basado en HTML5

Últimamente es una opción muy popular, especialmente si se conoce el mundo web, el utilizar alguna herramienta basada en HTML que genere aplicaciones para todas las plataformas.

Aunque existen muchas, quizás la más conocida es PhoneGap/Apache Cordova. Además, en las apps escritas en HTML5 y compiladas con estas herramientas podemos utilizar un *skin* concreto para que el aspecto sea lo más similar posible al de las aplicaciones nativas. Se puede obtener gran parte de la funcionalidad nativa del dispositivo móvil a través de librerías JavaScript. Evidentemente, estas facilidades tienen un precio, y es que las aplicaciones desarrolladas por este sistema no tienen el mismo rendimiento que una APP nativa; tampoco se tendrá acceso a todas las API nativas de cada plataforma, aunque sí a las más importantes.



Investiga

Últimamente se habla de reemplazar las apps nativas por las *Progressive Web Apps*, una generación de aplicaciones que bordean la frontera entre aplicaciones web y apps móviles tradicionales, combinando las ventajas de ambas, pero ¿sabes en qué consisten?

Como conclusión de este apartado, y para facilitar la elección, es recomendable que si se va a programar tan solo en una plataforma y no se conoce C# ni HTML, se aprendan las herramientas y el lenguaje nativo: Objective-C o Swift y Cocoa Touch para iOS, Java y el SDK de Android para Android, y C#/XAML para Windows Phone.

En caso de querer realizar aplicaciones para todas las plataformas, según los conocimientos de C# o HTML5, se puede optar por una solución con Xamarin o PhoneGap. Se puede concretar la cuestión con los conocimientos de los que se parte y las plataformas en las que se quiere programar.

1.5. Entornos integrados de desarrollo de aplicaciones móviles

Como es fácil imaginar si se opta por la programación nativa, es aconsejable familiarizarse con las herramientas de desarrollo propias de esa plataforma. En este apartado se hará un breve recorrido por los lenguajes e IDE que es aconsejable conocer en cada caso.

 PARA SABER MÁS

AWS Amplify es una biblioteca JavaScript que consta de un marco de desarrollo y servicios, de código abierto, que ofrece una forma rápida y sencilla de compilar aplicaciones móviles (iOS, Android) y web. Las herramientas para desarrolladores incluyen la consola de AWS Amplify, que permite compilar, implementar y alojar aplicaciones web, y probar aplicaciones móviles en dispositivos reales de iOS y Android.

1.5.1. Symbian

Para programar aplicaciones en Symbian no es necesario ningún conocimiento específico de un código único para este sistema, ya que es posible programar en él a partir de lenguajes como Java, C++ Visual Basic, Python, Perl, Flash Lite.

Origo IDE es un entorno de desarrollo apropiado para la programación de móviles Symbian de última generación, que incluye un editor de código, compilador y emulador, basándose en un sencillo lenguaje de script que proporciona resultados similares al uso del C++ nativo de Symbian. Tiene el inconveniente de que es una herramienta propietaria del alto costo, aunque tiene algunas versiones demo.

1.5.2. BlackBerry OS

BlackBerry Java Development Environment es un entorno completamente integrado de desarrollo y simulación para crear BlackBerry® Java Application para dispositivos BlackBerry. Gracias a BlackBerry JDE, puedes crear aplicaciones con el lenguaje de programación Java® ME y las API extendidas de Java para BlackBerry. Este IDE incluye herramientas de edición y depuración optimizadas para el desarrollo de BlackBerry Java Application.

1.5.3. Windows Phone

Esta plataforma soporta los lenguajes C# y Visual Basic.NET. Para el diseño se utiliza el lenguaje Silverlight (también conocido como XAML), y para videojuegos se puede utilizar XNA, que genera gráficos 2D/3D.

Visual Studio 2010 es una herramienta adecuada para programar aplicaciones para esta plataforma. Se necesita tener instalado Windows Phone SDK 7.1 que suministra API y otras herramientas.

1.5.4. iOS

Swift es un innovador lenguaje de programación para Cocoa y Cocoa Touch, con mejoras sobre Objective-C, que es mucho menos transigente con los errores y una sintaxis más complicada. La combinación de Swift con Xcode facilita y agiliza el desarrollo para esta plataforma, además de poseer un intuitivo interfaz (Interface Builder, una herencia de NeXT) y buenas herramientas de depuración.

1.5.5. Android

Java es el lenguaje nativo que usa Android. Las aplicaciones que hacen uso del hardware y se comuniquen con el sistema operativo, usarán este código. Es un lenguaje multiplataforma útil para el desarrollo de apps móviles y Desktop; es robusto, orientado a objetos, de arquitectura neutral y con la gran ventaja de las numerosas librerías que puedes encontrar en los repositorios.

Paralelo a Java se ha de conocer XML, un lenguaje de marcas similar a HTML. Es una especificación de recomendación W3C como lenguaje de marcado de propósito general. Será muy útil su conocimiento para crear las vistas de las aplicaciones y gestionar los recursos.

Como herramienta de desarrollo, tiene variadas opciones, pero quizás la más adecuada sea Android Studio, un IDE basado en el software IntelliJ IDEA de JetBrains y lanzado para reemplazar a Eclipse como el IDE oficial en el desarrollo de aplicaciones para Android. Es una herramienta completa para el desarrollo y la depuración de aplicaciones Android, con un sistema flexible de compilación y despliegue, además de muchas otras utilidades que facilitan la prueba de apps en variados dispositivos mediante su herramienta AVD (Android Virtual Devices).



PARA SABER MÁS

Google está realizando una fuerte apuesta por Kotlin un lenguaje de programación desarrollado por JetBrains cuya sintaxis es más limpia que Java, con menos necesidad de código y que permite la coexistencia simultánea con Java.

Figura 1.4
Kotlin.



Resumen

- Las características fundamentales de los dispositivos móviles van a depender de la capacidad de procesamiento, tamaño, portabilidad y conectividad.
- Las categorías en las que podemos encuadrar los dispositivos móviles según los estándares de la T38 y DuPont Global Mobility Innovation Team son Limited Data Mobile Device (Dispositivo Móvil de Datos Limitados), Basic Data Mobile Device (Dispositivo Móvil de Datos Básicos) y Enhanced Data Mobile Device (Dispositivo Móvil de Datos Mejorados).
- Los tipos de dispositivos móviles que puedes encontrar en el mercado son: teléfonos móviles, handhelds, netbooks, E-Book Readers, tablets, PDAs, smartphones, gadgets.
- Las comunicaciones móviles se encuadran en generaciones, iniciadas por la 0 (ondas de radio), hasta la actual 5G, que puede llegar a 1 Gbps de velocidad de transmisión.
- Los terminales móviles han ido evolucionando con el tiempo, reduciendo el tamaño, aumentando la capacidad de cálculo, prestaciones, conectividad... En la actualidad, se ha llegado a un mercado en el que, además de los grandes fabricantes tradicionales (Apple y Samsung), se están incorporando nuevas empresas altamente competitivas en el sector.

- La programación de dispositivos móviles está sujeta a ciertas limitaciones, como son las variadas plataformas, características del hardware, tamaño de pantalla y conectividad continua no asegurada, entre otras.
- Aunque la mayoría de los dispositivos móviles existentes en el mercado usan con mayor frecuencia Android e iOS, y en menor cantidad Windows Phone, como sistema operativo, otros que también es interesante conocer son BlackBerry OS, Symbian, Palm OS P (WebOS), Firefox OS y Ubuntu Touch.
- Casi todos los sistemas operativos para móviles tienen una arquitectura constituida en capas, donde el kernel realiza las principales tareas de enlace con el hardware del dispositivo.
- A la hora de programar en dispositivos móviles, puedes optar por realizar desarrollo nativo, desarrollo multiplataforma compilado o desarrollo multiplataforma basado en HTML5.
- Todos los sistemas operativos para móviles tienen un lenguaje de programación idóneo para desarrollar aplicaciones para ellos, siendo los más populares Android y Swift. Asimismo, estos requieren de entornos de desarrollo integrados y otras herramientas para facilitar las tareas del programador.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de las siguientes no es una característica propia de los Limited Data Mobile Device?:
 - a) Pantalla pequeña.
 - b) GPS.
 - c) Servicio de mensajes SMS.
 - d) Acceso Wap.
2. Cuando hablamos de un dispositivo con pantalla plegable sobre el teclado, utilizado fundamentalmente como agenda y con la posibilidad de ejecutar algunas aplicaciones, nos referimos a:
 - a) Netbooks.
 - b) E-Book Readers.
 - c) Personal Digital Assistant.
 - d) Handhelds.
3. ¿Qué generación de comunicación móvil está basado en el UTMS (Universal Mobile Telecommunications System)?:
 - a) Primera generación.
 - b) Segunda generación.
 - c) Tercera generación.
 - d) Cuarta generación.

4. ¿Qué móvil es considerado el primero en salir al mercado con formato *clamshell*?:

- a) Motorola Startac.
- b) Nokia 9000i.
- c) Kyocera QCP6035.
- d) Ericsson W810i.

5. ¿Cuál de las siguientes limitaciones tendrás que tener en cuenta a la hora de elegir el lenguaje con el que vas a desarrollar tu aplicación?:

- a) Sistema operativo.
- b) Velocidad de procesado.
- c) Tamaño de pantalla.
- d) Conectividad a la red.

6. En una arquitectura por capas, ¿cuál de las siguientes ofrece servicios como mensajería, comunicaciones, multimedia?:

- a) El kernel.
- b) El middleware.
- c) El entorno de ejecución de aplicaciones.
- d) Las interfaces de usuario.

7. ¿Cuál de los siguientes sistemas operativos de móvil fue lanzado por la Open Handset Alliance (OHA)?:

- a) Android.
- b) iOS.
- c) Symbian.
- d) Windows Phone.

8. ¿Cómo se llama la capa para el desarrollo de aplicaciones iOS?:

- a) Runtime.
- b) Cocoa Touch.
- c) Microkernel.
- d) Core OS.

9. ¿Qué es Xamarin?:

- a) Un lenguaje de desarrollo nativo.
- b) Una herramienta utilizada para integrar HTML5 en dispositivos móviles.
- c) Una herramienta basada en C# que permite compilar en nativo.
- d) Una herramienta para integrar páginas web en el móvil.

10. ¿Cuál de los siguientes entornos de desarrollo consideras más apropiado para programar Swift?:

- a) Visual Studio 2010.
- b) Xcode.
- c) Origo IDE.
- d) Android Studio.

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d

2.1. Introducción

Para el desarrollo nativo de aplicaciones en Android no basta con ser un buen conocedor del código (básicamente Java) y tener buena pericia en fundamentos de programación. La peculiaridad de los dispositivos móviles, la variabilidad que entre estos existen (como ya se estudió en el capítulo anterior) y, sobre todo, la multiplicidad de recursos que son capaces de aportar estos dispositivos, hacen necesario que antes de iniciar las tareas de codificación se conozcan los fundamentos y la estructura de componentes en los que se basa una aplicación en Android.

Además, el correcto manejo de algunos de estos componentes (Android Manifest, clase R, permisos...) es importante si se desea que las aplicaciones desarrolladas tengan alguna proyección de mayor alcance que el límite que pueda imponer el propio equipo informático y las herramientas de desarrollo.

En este capítulo se asentarán fundamentos de obligado conocimiento para que el avance en los próximos no encuentre mayor dificultad que la del aprendizaje del código.

2.2. Fundamentos de una aplicación en Android

Como ya se ha visto, Android es un sistema operativo multiusuario basado en Linux, donde de cada aplicación es considerada propiedad de un usuario distinto. El sistema asigna un identificador de usuario diferente a cada aplicación, por lo que los archivos incluidos en cada una tendrán permisos solo para ese usuario y solo él podrá tener acceso a ellos.

Esto hace que cada aplicación tenga su propio sistema de seguridad, con usuarios diferentes para cada una de ellas, permisos propios para cada usuario con máquinas virtuales propias y un proceso de Linux propio. De tal manera que Android utiliza el *principio de menor privilegio*, dando los permisos justos a cada aplicación. Todo ello lo convierte en un sistema operativo seguro.

No obstante, una aplicación puede solicitar permisos para acceder a datos (agenda), recursos (SD, Bluetooth) o funcionalidades (SMS, cámara) del dispositivo. Estos permisos asociados a la aplicación se conceden en la instalación de la misma o en el momento de usarlos (para versiones más modernas de Android), y los otorga el usuario.

Las aplicaciones Android están escritas en Java, un lenguaje de programación orientado a objetos, apoyado en determinados aspectos por XML. El SDK (kit de desarrollo de software) de Android compila el código, datos y recursos, incluyéndolo todo en un fichero APK.



Figura 2.1
Paso de APP a APK.

2.3. Componentes de una aplicación

Los componentes de una aplicación son bloques de código mediante los cuales el sistema puede relacionarse con esta. Cada componente tiene una identidad propia y cumple un papel específico.

Hay cuatro tipos distintos de componentes; cada uno tiene un fin específico y un ciclo de vida diferente (más adelante, serán estudiados con mayor profundidad). Para entender los fundamentos de una aplicación en Android, los primeros componentes con los que es necesario familiarizarse son los siguientes:

- a) *Activity*: es el principal componente de una aplicación de Android y se encarga de gestionar gran parte de las interacciones con el usuario. Representa una pantalla independiente con una interfaz de usuario. Una aplicación puede tener varias actividades que, aunque independientes, colaboran entre sí en el funcionamiento de la aplicación.
- b) *Service*: son aplicaciones que corren de fondo para hacer operaciones de larga duración o trabajo en procesos remotos. Un servicio no tiene interfaz de usuario. Una actividad puede iniciar el servicio y permitir que se ejecute o enlazarse con él para desarrollar su cometido.
- c) *Content Provider*: se ocupa de gestionar un conjunto de datos de una aplicación para compartir. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar información (si el proveedor lo permite).
- d) *Broadcast Receiver*: es una utilidad de Android que permite responder a anuncios broadcast (difusión) del sistema. Frecuentemente, un receptor de mensajes es simplemente un enlace con otros componentes realizando una cantidad mínima de trabajo.

A lo largo de este libro, se trabajará cada uno de estos componentes y se verá que para activarlos e iniciar su ciclo de vida, desde otro elemento de la aplicación e incluso desde una aplicación distinta, se hará mediante *Intent*, un elemento esencial que utiliza Android para moverse de una pantalla (Activity) a otra.

RECUERDA

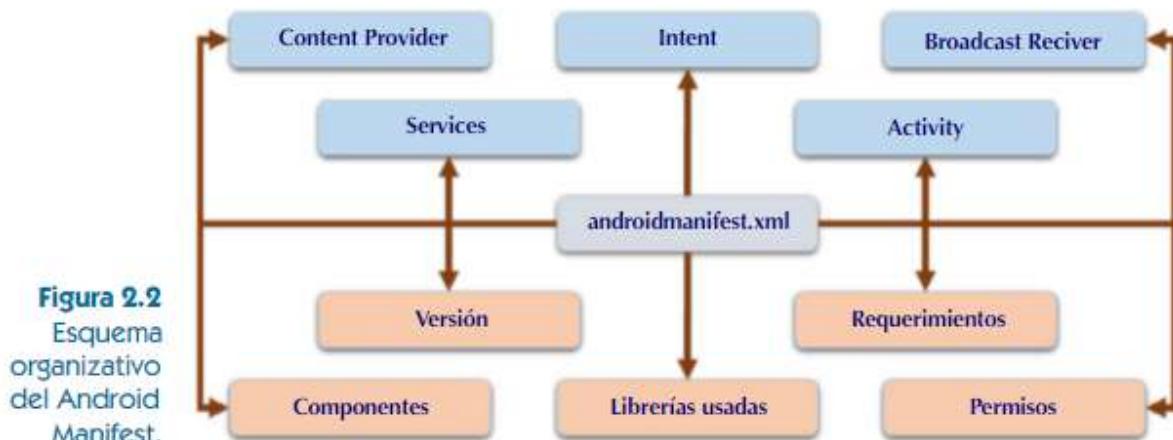
- ✓ Las aplicaciones Android están compuestas por uno o más componentes de aplicación (actividades, servicios, proveedores de contenido y emisores de notificaciones).

2.4. Android Manifest

Para incluir algunos de los anteriores componentes en las aplicaciones, será necesario que el sistema reconozca la existencia de ese componente, y eso lo hará leyendo el archivo Manifest (AndroidManifest.xml) de la APP. El archivo de manifiesto proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la APP. Todas las aplicaciones tienen este archivo, cuya gestión será fácil a

través del entorno de desarrollo; una aplicación debe tener declarados todos sus componentes en este archivo.

La principal tarea es informar al sistema acerca de los componentes de la aplicación. Otras funciones de este archivo serán registrar los permisos asociados a la aplicación, librerías que utiliza, declarar las necesidades de software/hardware, así como el nivel de API mínimo que requiere la aplicación.



Este archivo está escrito en lenguaje de marcas y tiene un aspecto similar al que se puede ver a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="es.libro.ejemplo">
    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".Fragmentos">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Aunque existen otras (que, dependiendo del apartado que se trabaje, se irán viendo), las etiquetas que es necesario conocer inicialmente son las siguientes:

- *Manifest*. Engloba a las demás etiquetas. Define el espacio de nombres, el nombre del paquete y atributos del mismo.
- *Application*. Contiene metadatos de la aplicación (título, icono, tema), etiquetas de actividades, servicios, proveedores de contenidos y receptores de broadcast.
- *Uses-Sdk*. Indica las versiones del SDK sobre las que podrá ejecutarse, el nivel mínimo de API y el utilizado para su desarrollo.

- *Uses-Permission.* Declara los permisos que la aplicación necesita para operar. Serán presentados al usuario durante la instalación para que los acepte o deniegue. Algunos de los permisos que podemos declarar aquí son los siguientes:
 - INTERNET: conexión a Internet.
 - READ_CONTACTS: leer en la lista de contactos.
 - WRITE_CONTACTS: escribir en la lista de contactos.
 - SEND_SMS: enviar SMS.
 - ACCESS_COARSE_LOCATION: localización mediante telefonía.
 - ACCESS_FINE_LOCATION: localización mediante GPS.
 - BLUETOOTH: permite el uso del Bluetooth.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Actividad propuesta 2.1



Busca en la página de Android Developer qué utilidad puede tener la inclusión del siguiente código en el Manifest:

```
<uses-permission android:name = "permiso" android:maxSdkVersion = "entero"/>
```

- *Permission.* Define un permiso que se requiere para que otras aplicaciones puedan acceder a partes restringidas de la aplicación.
- *Instrumentation.* Nos permite definir un test de ejecución para las actividades y servicios.



PARA SABER MÁS

La administración de permisos en Android es un aspecto muy cuidado por Google y que ha ido evolucionando con las diferentes versiones de Android, especialmente desde Android 6.0 Marshmallow. En un apartado específico de este capítulo se estudiará la gestión de permisos un poco más a fondo.

2.5. Recursos de una aplicación

Este término hace referencia a los datos que utiliza la aplicación, entendiendo como tal las imágenes, textos, estilos. Estos, junto con el código, configuran una aplicación.

Los recursos pueden estar predeterminados para la aplicación que se haya programado y se usarán siempre sin importar el tipo o la configuración del dispositivo sobre el que se vaya a ejecutar. O bien pueden crearse recursos alternativos, que son aquellos que se diseñan para usar con una configuración específica de dispositivo (tablet, pantallas grandes...), colaborando, en parte, con la responsividad de la aplicación.

TEN EN CUENTA

- ✓ Las aplicaciones elaboradas se ejecutarán en dispositivos con diferentes características y configuraciones. Se deberán tener en cuenta, especialmente, los apartados de densidad de pantalla (usando recursos gráficos -ldpi, -mdpi, -hdpi, según resolución), idiomas del usuario (almacenando los recursos de string en directorios values-es, values-it), orientación y tabletas (dando soporte -sw600dp, -sw720dp), entre otros.

Además, los recursos pueden estar gestionados por ficheros XML, deben tener asignado un identificador (id) y suelen estar alojados en una subcarpeta dentro de la carpeta principal denominada *res*. Los nombres de estas subcarpetas son limitados y predefinidos por Android.

TOMA NOTA



Nunca se deben guardar archivos de recursos directamente dentro del directorio *res*/ ya que se produciría un error en la compilación.

En el listado que se muestra a continuación se relacionan las más frecuentes y el tipo de recurso que pueden contener:

- *Carpetas res/drawable*: recursos dibujables, ficheros de bitmaps o de imágenes “escalables”. También pueden definirse aquí formas y colores de objetos dibujados.
- *Carpetas res/layout*: definidos como ficheros XML. Engloban los elementos visuales que definen el interfaz de nuestra aplicación.
- *Carpetas res/animator*: permite crear animaciones sencillas sobre uno o varios gráficos, como rotaciones, fading, movimiento y estiramiento. Cada animación se define en un fichero XML.
- *Carpetas res/mipmap*: con contenido similar a drawable, el directorio mipmap alberga elementos bitmap, aunque se suele utilizar específicamente para ubicar el icono de la aplicación.
- *Carpetas res/menu*: son ficheros XML donde se definen las diferentes opciones de los menús, submenús, barras de navegación incluidos en las aplicaciones.
- *Carpetas res/values*: es una carpeta con carácter genérico con ficheros XML que configuran diferentes aspectos de la aplicación, como es el color, dimensiones, cadenas de texto, estilos.

RECUERDA

- ✓ Un *estilo* es el conjunto de propiedades (tamaño, relleno, color...) que definen la apariencia de un objeto visual. Un *tema* es un estilo que se aplica a toda una *Activity* o a la aplicación es su conjunto. Si un estilo se usa como un tema, se verán implicadas todas las vistas de la actividad o de la aplicación.

- *Carpeta res/xml:* almacena archivos XML utilizados por Android para dar soporte a tareas múltiples, como la que permite configurar búsquedas.
- *Carpeta res/raw:* se usa para almacenar los archivos raw de la aplicación (audio y vídeo). Este directorio tiene ciertas limitaciones en cuanto al nombre de archivos que puede tener (mayúsculas, números, caracteres especiales), por lo que, si se necesita usar el nombre real del archivo, entonces se deben colocar los archivos raw en el directorio asset de Android.



Investiga

Accede a la página sobre tópicos (dedicada a recursos) en Android Developer a través de este QR.

Investiga en ella cómo debe hacerse la provisión de recursos y la agrupación de los mismos. Observa la importancia del uso de recursos alternativos que permiten admitir configuraciones de dispositivos con características propias y específicas.



2.5.1. Cómo acceder a los recursos

Como se ha visto anteriormente, todos los recursos de tu aplicación tienen asignado un identificador (generalmente, Android lo hace de manera automática), por lo que quedan el tipo de recurso y su id registrados en el archivo de la clase R de la aplicación. Esto puede ser utilizado para acceder al recurso desde el código Java o XML de la manera que a continuación se expone.

Para acceder a los recursos en XML se debe escribir el nombre del recurso y la dirección del archivo donde este se encuentra. En el ejemplo siguiente, se ha definido en ficheros XML el color, el estilo y el contenido de la cadena de texto. Esto hace muy cómodo el realizar variaciones de las propiedades de este objeto (o aplicárselas a otros) sin tener que retocar el código Java de la aplicación.

```
<TextView
    android:id="@+id/texto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/colorPrimario"
    android:text="@string/titulo"
    style="@style/EstiloTitulo"/>
```

En Java, para acceder a un recurso (getResources), se ha de localizar este a través de la clase R, al igual que en XML, indicando directorio (o fichero XML) y nombre del recurso.

```
getResources().getColor(R.color.colorPrimario);
getResources().getDrawable(R.drawable.dibujo);
miImagenView.setImageResource(R.drawable.imagen);
```

Investiga

Como se irá viendo, Android nos facilita multitud de recursos, que muchas veces costará trabajo localizar o identificar. Para acceder a ellos, se debe hacer de manera similar a la que ya se ha visto, pero anteponiendo al recurso la palabra *android*.

Aquí se muestra un ejemplo de cómo acceder al color negro que nos proporciona Android, tanto desde XML como desde Java.

```
    android:textColor="@android:color/black"
    getResources().getColor(android.R.color.black);
```

2.6. La clase R

Se trata de una clase que contiene variables estáticas en las que se identifica cada tipo de recurso. Android lee el fichero XML, carga todas las estructuras solicitadas en memoria y mantiene el fichero R como referencia directa a los recursos cargados. Por tanto, cada atributo tiene una dirección de memoria asociada referenciada a un recurso en específico. En la figura 2.3, *mensaje* contiene la cadena “Hola Mundo”, que está almacenado en la dirección de memoria 0x7f0d0028.



Figura 2.3
Organización de la clase R.

Aunque Android Studio ha mejorado mucho la estabilidad de la clase R, en más de una ocasión se presentan errores en esta, que impedirán, mientras no se solucionen, el seguir avanzando en el desarrollo de la aplicación. Y aunque no existe una pauta que seguir para reparar los errores en la clase R, es importante asegurarse que no existan enlaces rotos, es decir, que no exista ningún error en los archivos XML, y si todo aparenta estar correcto, se pueden utilizar herramientas del entorno de desarrollo, como Build/Clean Project o Build/Rebuild Project.



TOMA NOTA

Nunca debe modificarse el archivo R.java manualmente. Este se genera a través de la herramienta AAPT (Android Asset Packaging Tool) cuando se compila el proyecto, por tanto, todos los cambios que se hagan manualmente se anularán tras cada compilación.

2.7. Estructura de un proyecto

Hasta el momento, se han visto los fundamentos y elementos en los que se basa una aplicación desarrollada en Android; ahora debemos de organizarlos y estructurarlos dentro del proyecto. Afortunadamente, toda esa labor recaerá sobre la herramienta de desarrollo utilizada (en nuestro caso, Android Studio), que a través de una serie de preguntas o decisiones iniciales nos dejará un entorno preparado para que iniciemos las labores de codificación.

Para poder entender de manera correcta cómo se estructura un proyecto en Android, y para poder iniciarnos en la práctica de código, se ha de tener instalado el entorno de desarrollo con el que se vaya a trabajar. Actualmente, el proceso es extraordinariamente simple, en especial si se utiliza Android Studio (lo que se hará en este libro), ya que Google ha simplificado todas las tareas de instalación mediante un Bundle (paquete) que se ocupa de gestionar todo el proceso, y tan solo recurre al usuario para algunas cuestiones básicas de ubicación y configuración.

Por otro lado, el proceso tiene algunas variaciones, dependiendo del sistema operativo utilizado y, en su caso, de la versión del mismo; aquí veremos algunas nociones básicas de la instalación, comunes a cualquiera de las variantes utilizadas, dejando para otros módulos (Entornos de Desarrollo) la profundización en cada uno de los aspectos de esta herramienta.

2.7.1. Instalación de Android Studio

Este entorno de desarrollo integrado presentado en 2013 para la plataforma Android reemplazó a Eclipse como IDE oficial para el desarrollo de aplicaciones para Android. Se basa en IntelliJ IDEA de JetBrains, es de licencia gratuita y puede obtenerse para cualquier sistema operativo. La última versión estable es la 3.4.2, de abril de 2019.

Aunque ha mejorado mucho, es un IDE con gran necesidad de requisitos del sistema. Para la versión 3 se necesita un mínimo de 3GB de RAM (8GB recomendado), además de 1 GB adicional si se utiliza el emulador. En cuanto al espacio en disco, se debe contar al menos con 4GB (IDE, SDK, emulador, caches) y, lógicamente, tener instalado el Java Development Kit (JDK).

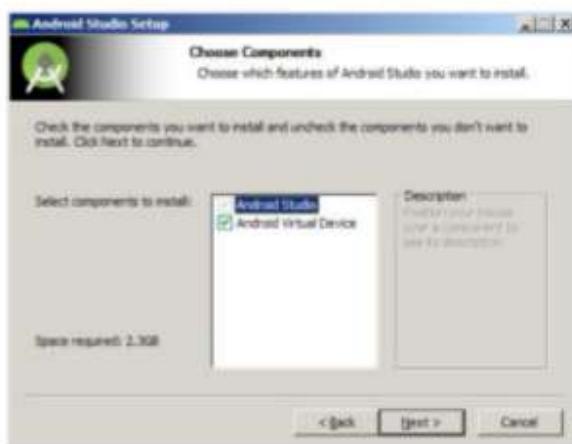


Figura 2.4
Elección de componentes.

Su instalación es sumamente sencilla; la descarga se puede hacer de la web oficial de Android (<http://bit.ly/2TLAYro>). El proceso de instalación puede tener pequeñas modificaciones, según la versión de Android Studio que se vaya a instalar: en la versión 3.5 de agosto de 2019, una vez lanzada la instalación tan solo hay que seguir una serie de pantallas que, en su mayoría, solo habrá que pulsar siguiente (next), como son la de aceptar términos, bienvenida, componentes que se van a instalar. En esta, además del entorno, puede marcarse la instalación del emulador AVD (AndroidVirtual Device), que aunque no es obligatorio, sí es recomendable, pese a que se tenga pensado realizar el desarrollo sobre dispositivos reales.

También preguntará dónde ubicar el entorno. Se debe poner especial atención en este apartado y procurar ubicarlo en directorios y unidades fácilmente accesibles (nunca utilizar caracteres “extraños”) y, a ser posible, en unidades especialmente rápidas. Una vez descargados los paquetes y terminada la instalación, se pasa al proceso de configuración, donde se nos consultará si buscar e importar configuraciones anteriores del entorno.

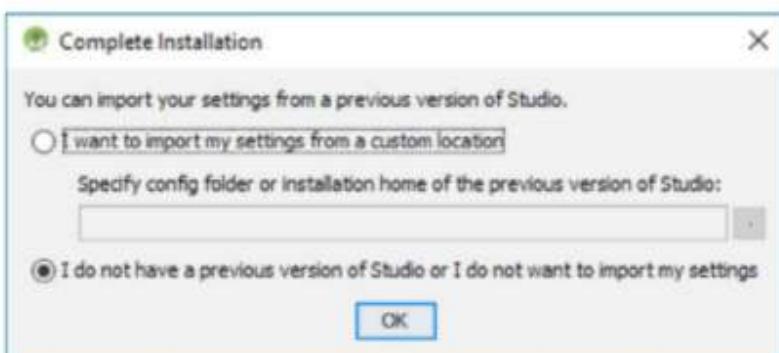


Figura 2.5
Importación
configuración.

También, tras la ventana de bienvenida, nos consultará sobre el tipo de dispositivo para el que se programará y el estilo (tema) con el que queremos que se muestre la interfaz de usuario.

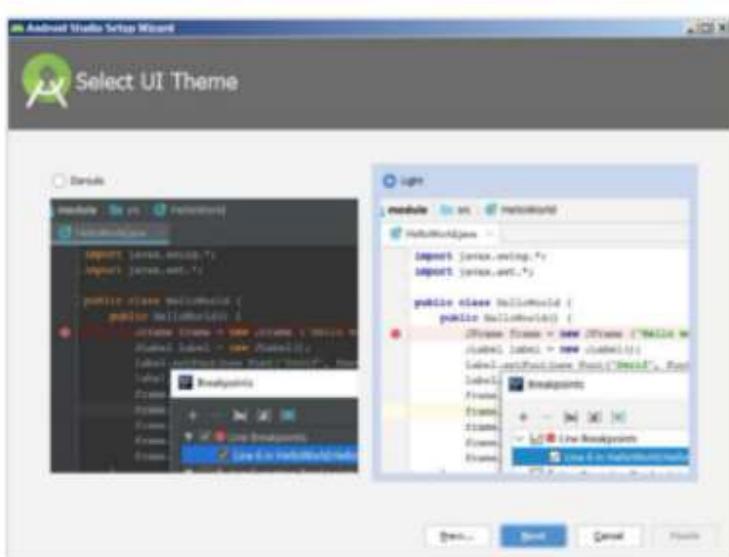


Figura 2.6
Selección de tema del IU.

Por último, hará una revisión y actualización de algunos componentes necesarios para las tareas de desarrollo, como son la SDK (platform), el Intel @ HAXM, o los AVD, entre otros.



SABÍAS QUE...

El Intel HAXM (Hardware Accelerated Execution Manager) es un sistema de virtualización que nos ayudará a mejorar el rendimiento del emulador de Android cuando se utilice este tipo de procesadores.

2.7.2. Iniciando el entorno de desarrollo

Terminada la instalación, el proceso de iniciar la primera aplicación consistirá en una serie de pasos que definirán algunas características generales del mismo.

En la primera pantalla preguntará el tipo de vista que se aplicará a la actividad principal. Aunque se puede elegir entre numerosas variantes, resulta aconsejable escoger en los primeros pasos del aprendizaje la denominada *Empty Activity*. En la siguiente pantalla habrá que dar un nombre a la aplicación (debe empezar por una letra en mayúscula), el dominio (Package) de la misma (importante a la hora de empaquetarla) y dónde se ubicará.

En el dominio es necesario separar las palabras por punto y normalmente se comienza colocando el prefijo “com” o “es”. Luego se pone cualquier nombre (el de la empresa o proyecto) y, por último, el nombre de la aplicación (es.libro.programa).

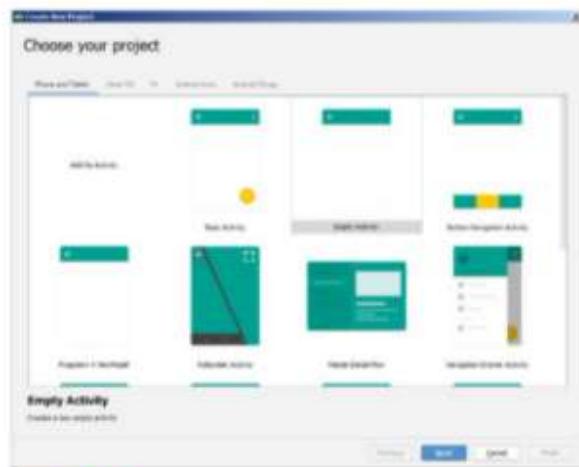


Figura 2.7
Selección de proyecto.

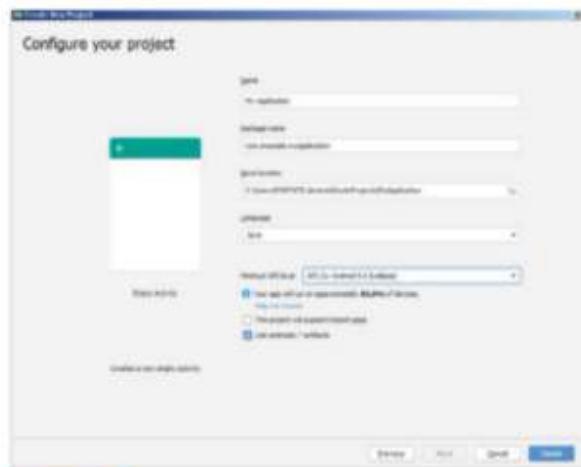


Figura 2.8
Configuración de proyecto.

Además, se ha de elegir lenguaje (Java o Kotlin). La mínima API que aceptará la aplicación (recomendable Lollipop o superior), pulsando el enlace “Help me choose” mostrará un gráfico con los dispositivos que aceptarán la aplicación en caso de subirla al Play Store.



Figura 2.9
Porcentaje de distribución según versiones de Android.

El activar Android Instant App Support permite lanzar las aplicaciones, en fase de desarrollo o prueba, disminuyendo los tiempos de espera. También se puede marcar el uso del complemento de Android X, que permite utilizar la biblioteca de esta versión.

RECUERDA

- ✓ Android está fundamentado en Java y, por tanto, se ha de tener instalado el JDK (kit de desarrollo de Java). Además, Android debe saber localizarlo mediante una correcta configuración de las variables del entorno. Lo normal es que estas estén automáticamente configuradas, pero, en caso contrario (Android Studio no reconocería Java), se deberán configurar a mano.

En Linux, etc/eniroment e incorporar:

```
JAVA_HOME="/usr/java/jdk1.8.0_25"
```

En Windows, dentro de propiedades del sistema:

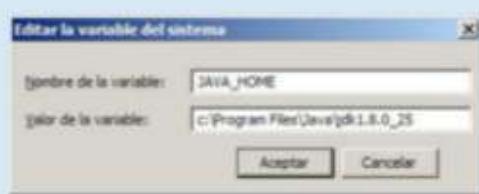


Figura 2.10
Edición de la variable del sistema.

Actividad propuesta 2.2



Inicializa un proyecto y calcula cuántos usuarios podrían acceder a la aplicación si tuvieran en sus dispositivos las siguientes versiones de Android: KitKat, Lollipop (Api 22), Nougat (Api 25) y Oreo.

2.7.3. Elementos de un proyecto

Ahora ya es posible escribir código. El entorno de desarrollo debe tener un aspecto muy similar al de la imagen siguiente. En este apartado, tan solo se repasará dónde se ubican los elementos que son necesarios a la hora de desarrollar una aplicación en Android.

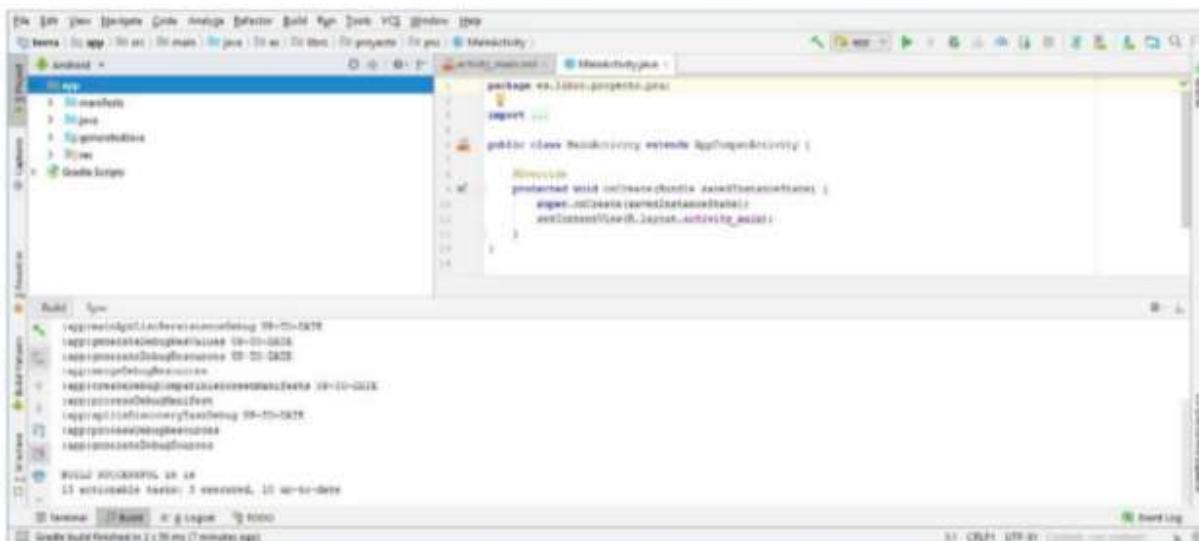


Figura 2.11
Interface de Android Studio.

Nos centraremos en el panel superior izquierdo (Project), ya que es muy importante familiarizarse con los elementos que contiene (debe mostrarse según los directorios de Android, lo que suele estar por defecto). Se deben mostrar dos carpetas principales, *app* y *Gradle Scripts*.

La primera (*app*) contiene cuatro subcarpetas: *manifest*, *java*, *generatedJava* y *res*. *Manifest* contiene el fichero *AndroidManifest.xml*, del que ya se ha hablado. En la carpeta *java* se incluyen los diferentes ficheros de la aplicación escritos en este lenguaje (si hubiera más de uno, estaría cada uno en su ruta) y *res* es la carpeta de recursos, dividida en la estructura de directorios anteriormente vista (generalmente, no aparecen todos). Por último, *generatedJava* es una carpeta que contiene todas las clases a partir de las cuales se construirá el proyecto, incluidas las bibliotecas y la clase *R*.

La segunda gran carpeta (*Gradle Scripts*) contiene la información necesaria para la compilación del proyecto, por ejemplo la versión del SDK de Android utilizada para compilar, la mínima versión de Android que soportará la aplicación, referencias a las librerías externas utilizadas, etc. El contenido de esta carpeta se modificará en alguna ocasión, especialmente cuando incorporemos librerías externas o de terceras partes.



Actividades propuestas

- 2.3.** Entra en los Setting de Android Studio (Ctrl+Alt+S) y comprueba los Platforms de Android que tienes instalados o parcialmente instalados y las SDK Tools instaladas.
- 2.4.** Inicia Android Studio y entra en el archivo build.gradle (Module:app), que se encuentra en la carpeta Grandle Scripts, y anota las dependencias que implementará tu entorno en las aplicaciones que desarrolles.

2.7.4. Configurar el dispositivo donde probar las aplicaciones

Como ya se ha comentado anteriormente, para probar el código en fase de desarrollo se necesita un dispositivo destino donde instalar la APK y visionar la aplicación. Para ello, se puede optar por dos posibilidades: por un lado, hacerlo en un emulador (puede ser el mismo que nos proporciona Android o utilizar otros externos), o bien utilizar un dispositivo físico real.

Siempre es recomendable esta segunda opción, ya que, además de ser más rápida y consumir menos recursos, se obtendrá una visión más real del resultado de la aplicación. Pero tiene el inconveniente de que difícilmente se podrá ver este resultado en variados dispositivos en cuanto a tamaño, versión de Android o características. Por lo anterior, sería recomendable una versión combinada: inicialmente probar en un dispositivo físico y, conforme la aplicación vaya adquiriendo complejidad, hacerlo en otros dispositivos emulados para ver qué tal responden a ella.

Para conectar un dispositivo físico a Android Studio es necesario que el sistema operativo lo reconozca y no dé conflictos. Además, en el dispositivo móvil han de estar habilitadas las opciones de desarrollador (muy importante). Para esto, cada marca tiene su sistema, pero generalmente suelen habilitarse dentro del apartado Información del Teléfono, pulsando varias veces alguna de las opciones que allí te encuentras (Versión de MIUI, Versión de Android...).

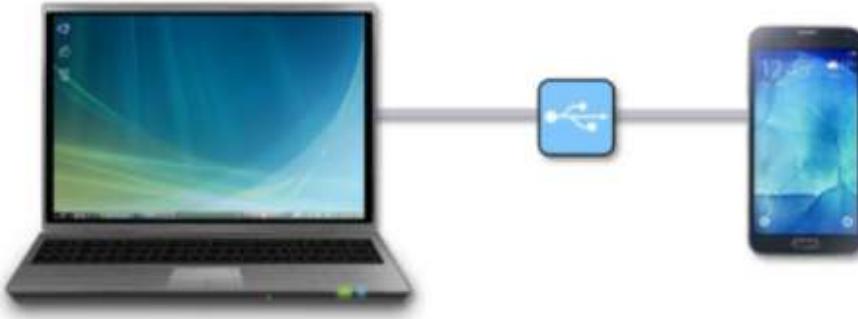


Figura 2.12

Conexión USB entre PC y dispositivo móvil.

Si se tiene esta opción habilitada, Android se comunicará con el móvil a través del ADB (Android Debug Bridge), pero es muy importante que el sistema operativo no muestre errores en el ADB Interface (necesario para el enlace USB). Esto suele ocurrir especialmente en Windows de versiones anteriores, cuando no están correctamente instalados los controladores ADB pertenecientes al modelo de móvil (en muchos casos, los generales de Google suelen ser suficientes). También, en caso de necesidad, puede recurrirse al Connection Assistant (asistente de conexión USB), que se encuentra dentro de la opción Tools.



Figura 2.13
Panel de Creación/Configuración de Android Virtual Devices (AVD).

Por otro lado, el emulador de Android Studio es bastante fácil de configurar, ya que dispone de un sencillo gestor (AVD Manager, también dentro de Tools). Este cuenta con algunos dispositivos preconfigurados y, además, es posible realizar una configuración totalmente a medida (incluso importarla), basta con definir los parámetros que habitualmente definen un dispositivo móvil.

Actividad propuesta 2.5



Crea mediante el AVD Manager un dispositivo virtual que tenga iguales o similares características que tu teléfono móvil habitual.

2.8. Los permisos en Android

Uno de los elementos importantes en los que ha ido trabajando Google, versión a versión, ha sido la seguridad en Android. Lo que se ha traducido, desde el punto de vista del usuario, en métodos de identificación cada vez más fiables, y en una restricción con respecto a lo que las aplicaciones podían hacer en nuestro móvil.

Como se ha visto, los permisos son declarados por el programador en el `AndroidManifest.xml`, de tal manera que los usuarios finales son capaces de saber qué va a poder y qué no va a poder hacer la aplicación. Tanto Play Store como Android son responsables de mostrar al usuario dichos permisos, incluso antes de descargar la aplicación en nuestro dispositivo, debiendo el usuario decidir el aceptar o no los permisos, lo que, en la mayoría de los casos (de no aceptarlos), supondría la ausencia de parte o toda la funcionalidad de la aplicación.

A partir de Android 6.0 Marshmallow existe un nuevo administrador de permisos, una medida de seguridad que nos dará más control y privacidad, ya que podemos revisar los permisos

de las aplicaciones de una forma muy sencilla e intuitiva. Al instalar una aplicación, nos pedirá autorización por cada permiso que vaya a necesitar, de tal manera que podemos rechazarlo en un momento determinado y autorizarlo con posterioridad, si así lo deseamos. Podemos, además, gestionarlos manualmente por aplicaciones o por tipo de permiso, de modo que aplicaciones podrán tener ahora permisos permanentes pero revocables, y Android nos informará mediante una notificación cuando una aplicación tenga algún permiso permanente y sensible, en el momento de ejecutarla.

Por todo ello, ahora, como programador, se debe saber que a la hora de codificar las aplicaciones no es suficiente con declarar los permisos más críticos en el Manifest, sino que también se debe solicitar acceso en el momento de su uso.

Si alguno de los permisos que se van a utilizar en la aplicación aparece en la lista antes citada, se ha de incluir un código que compruebe si el permiso ha sido concedido, lo que se hará mediante el método `ContextCompat.checkSelfPermission(thisActivity, Manifest.permission.CAMERA)`.

```
int permissionCheck=ContextCompat.checkSelfPermission(thisActivity, Manifest.permission.CAMERA);
```

Si tiene el permiso autorizado, el método muestra `PackageManager.PERMISSION_GRANTED`, y esta puede continuar con la operación. Si, por el contrario, no tiene el permiso, el método muestra `PERMISSION_DENIED`, y la aplicación debe solicitar permiso al usuario mediante el método `requestPermissions`. Después, se utilizará el método `onRequestPermissionsResult` para bifurcar nuestro código, según esté permitida la funcionalidad o no.

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_CAMERA: {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                //Código de permiso aceptado
            } else {
                //Código de permiso denegado
            }
            return;
        }
    }
}
```

Recursos web

- En el primer código QR puedes consultar en la página de Android Developer cuáles son los permisos considerados especialmente peligrosos.
- En el segundo código QR encontrarás un vídeo de la Universidad Politécnica de Valencia sobre los permisos en Android 6.0 Marshmallow.



Resumen

- Android es un sistema operativo multiusuario basado en Linux que utiliza el principio de menor privilegio.
- Android compila el código, datos y recursos, incluyéndolo todo en un fichero APK, que será el que se instale en el dispositivo móvil.
- Una aplicación de Android puede tener cuatro tipos diferentes de componentes: Activity, Service, Content Provider y Broadcast Receiver. Además, Intent es un elemento esencial, que es utilizado para movernos entre estos componentes.
- El Android Manifest es un archivo que proporciona información esencial sobre la aplicación al sistema Android, que le es necesaria para poder ejecutar el código de la APP.
- Los elementos auxiliares de una aplicación son calificados como recursos, y suelen tener asignado un identificador. Estos recursos son ubicados en carpetas predeterminadas por la estructura del proyecto, según el tipo y características de cada uno de ellos.
- La clase R es un importante fichero que contiene variables estáticas en las que se identifica cada tipo de recurso existente en la aplicación.
- Para desarrollar aplicaciones nativas en Android utilizaremos un entorno de desarrollo (Android Studio, en nuestro caso) debidamente instalado y configurado, además de un dispositivo móvil (real o emulado) sobre el que ejecutar las aplicaciones creadas.
- Android ha ido mejorando las medidas de seguridad frente a las aplicaciones mediante un nuevo administrador de permisos, que nos permite revisarlos de una forma sencilla e intuitiva.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de estos puede ser considerado como el componente principal de una aplicación?:
 a) Activity.
 b) Service.
 c) Content Provider.
 d) Broadcast Receiver.
2. ¿De cuál de estos elementos no aparece información en el Manifest?:
 a) Metadatos de la aplicación (ícono, título, tema...).
 b) Versión de la API sobre la que puede ejecutarse la aplicación.
 c) Tamaño de la pantalla para la que funciona la aplicación.
 d) Declaración de permisos necesarios.

3. ¿Cuál de estas carpetas no está incluida en los recursos de un proyecto?:

- a) Carpeta res/layout.
- b) Carpeta res/mipmap.
- c) Carpeta res/Grandle.
- d) Carpeta res/values.

4. ¿Mediante qué elemento se identifican los recursos registrados en la clase R?:

- a) Un puntero a memoria.
- b) Una clase.
- c) Un fichero XML.
- d) Un valor numérico.

5. ¿Cuándo has de modificar el fichero R.java?:

- a) Cada vez que incluyas un nuevo recurso.
- b) Siempre que cambia un recurso.
- c) Nunca.
- d) Antes de compilar la aplicación.

6. ¿En cuál de los siguientes entornos de desarrollo se fundamenta Android Studio?:

- a) NetBeans.
- b) IntelliJ IDEA.
- c) Eclipse.
- d) Visual Studio Code.

7. Cuando pongas un nombre a una aplicación:

- a) Es conveniente que todo sea en mayúsculas.
- b) Es conveniente que todo sea en minúsculas.
- c) Por motivos de seguridad, debe tener letras, números y signos de puntuación.
- d) Es conveniente que tan solo sea mayúscula la primera letra.

8. ¿En cuál de estas carpetas debes hacer referencia a las librerías externas cuando las utilices?:

- a) java.
- b) generatedJava.
- c) Res.
- d) Grandle Scripts.

9. ¿Mediante qué herramienta se comunica Android Studio con el dispositivo móvil?:

- a) AVD.
- b) ADB.
- c) APK.
- d) APP.

10. ¿Con qué versión de Android se produce un importante cambio en la gestión de permisos?:

- a) Jelly Bean.
- b) Lollipop.
- c) Marshmallow.
- d) Oreo.

SOLUCIONES:

1. **a** **b** **c** **d**

2. **a** **b** **c** **d**

3. **a** **b** **c** **d**

4. **a** **b** **c** **d**

5. **a** **b** **c** **d**

6. **a** **b** **c** **d**

7. **a** **b** **c** **d**

8. **a** **b** **c** **d**

9. **a** **b** **c** **d**

10. **a** **b** **c** **d**

Mapa conceptual



Glosario

Aplicaciones responsivas. Son aquellas que se adaptan a diferentes tamaños y proporciones de pantalla mediante un cambio o reorganizando del contenido en el interface, que puede suponer el escalado de imágenes, cambio de orden o posición en los menús.

Broadcast. En Android, la difusión amplia, difusión ancha o broadcast es la transmisión de datos que serán recibidos por todos los elementos que se encuentren activos y operativos en el dispositivo móvil.

Bundle. Son sistemas de almacenaje (paquetes) que se usan para pasar información entre diferentes actividades de Android.

Grandle Scripts. Carpeta de un proyecto de Android que contiene información necesaria para la compilación del proyecto.

Intent. Importante elemento de Android que es utilizado para cambiar de una pantalla o actividad a otra.

Kit de desarrollo de software (SDK). Se denomina así al conjunto de herramientas de desarrollo de software que permite al programador crear una aplicación informática para un sistema concreto.

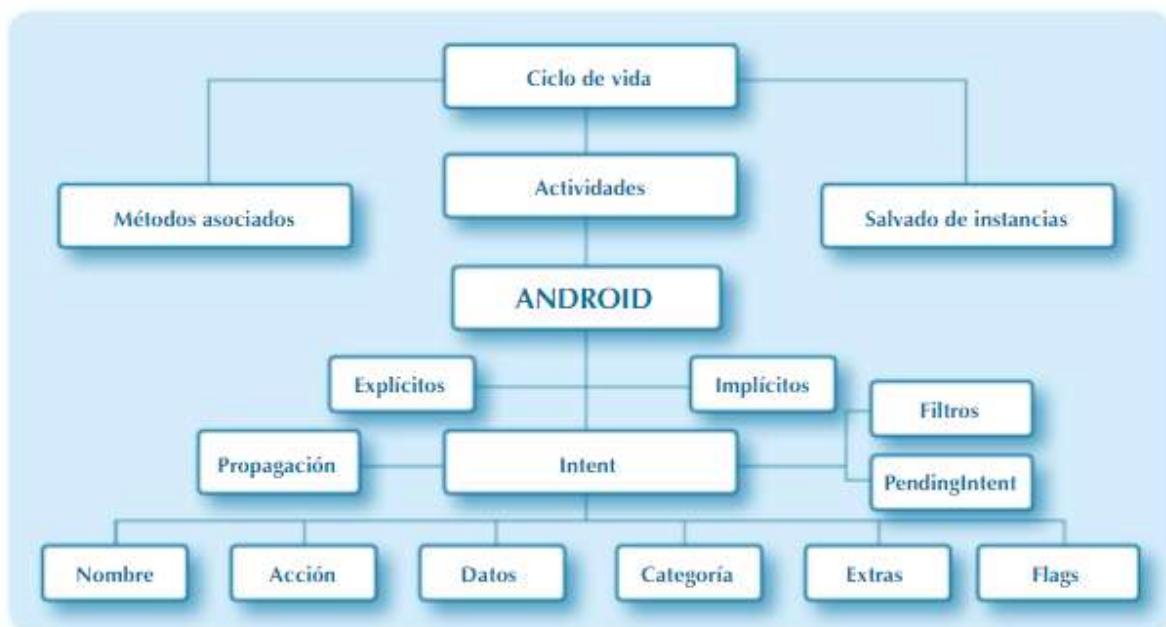
Platform. Término que se aplica generalmente en sistemas operativos a una o varias tecnologías que se utilizan como base sobre la que desarrollar aplicaciones, establecer procesos o implementar nuevas tecnologías.

Las actividades en Android

Objetivos

- ✓ Conocer la clase Activity.
- ✓ Comprender el ciclo de vida de una aplicación.
- ✓ Familiarizarse con el funcionamiento de la pila (Back Stack) en Android.
- ✓ Identificar y manejar los métodos asociados al ciclo de vida de una actividad.
- ✓ Crear la primera aplicación en Android.
- ✓ Valorar el uso de las librerías en Android.
- ✓ Facilitar el salvado de instancias y parámetros de las actividades.
- ✓ Conocer las intenciones (Intent), y trabajar con ellas y con algunos de sus filtros.

Mapa conceptual



Glosario

API (*Application Programming Interface*). Conjunto de protocolos, funciones y comandos informáticos que permiten a los desarrolladores crear programas específicos para determinados sistemas operativos y, además, integrar el software de las aplicaciones.

Callback (*llamada de vuelta*). Es una función que recibe parámetros de una función primaria, que se ejecutará una vez que esta haya terminado de ejecutarse, realizando algún tipo de actividad según la información recibida.

Foreground. En términos informáticos, es contrario a **Background**, y hace alusión a aquellos procesos que se ejecutan en primer plano o que están mostrándose a través de la pantalla.

Layout. Define la estructura visual en la interfaz de usuario; actúa como contenedor de objetos. Puede declararse en Android mediante XML o añadiendo una instancia en Java en tiempo de ejecución.

Lenguaje de marcas. Son aquellos lenguajes o formas de codificación en los que, junto al texto y el código, incorporan etiquetas o marcas que dan información suplementaria sobre la estructura del texto o su presentación.

Override. Término utilizado en Java (como lenguaje orientado a objetos) que permite sobrecargar o modificar el comportamiento original de un método heredado.

URI (*Uniform Resource Identifier*). Es un estándar definido que permite identificar y facilitar la localización de recursos en Internet (páginas web, imágenes, videos...).

3.1. Introducción

En este capítulo se darán los primeros pasos en programación nativa en Android, para lo que es fundamental conocer las Activity (actividades) y, especialmente, su ciclo de vida y los métodos asociados a cada uno de sus momentos. También se estudiará el funcionamiento la pila y, lo que es más importante, cómo reaparecen las tareas, conservándose las instancias de manera idéntica a como estaban antes de que pasaran a un segundo plano.

En el segundo bloque se abordarán las intenciones (Intent), una potente clase que se ocupa de llamar a una u otra actividad, de manera directa (explícita) o indirecta (implícita), dejando que sea el sistema quien nos aconseje cuál de las utilidades o aplicaciones en él instaladas es más útil para la acción que deseamos realizar.

3.2. Las actividades

La actividad (Activity) es el principal componente de una aplicación de Android y se encarga de gestionar gran parte de las interacciones con el usuario. A nivel de lenguaje de programación, una actividad hereda de la clase Activity y se traduce en una pantalla. Por tanto, y en términos generales, una aplicación de Android tendrá tantas Activities como pantallas distintas tenga.

Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. En el diseño de una aplicación, las Activities tendrán una secuencia de aparición según la lógica con la que se haya construido la misma. Aparecerán una u otras pantallas según el flujo del programa y las decisiones que en cada caso tome el usuario.

Desde una pantalla principal (main), se podrán realizar desplazamientos de una a otra Activity y, conforme se va “navegando” por las diferentes pantallas de la aplicación, el dispositivo va recordando la secuencia “visitada”, de tal manera que si se retrocede utilizando los botones del dispositivo o la lógica de la aplicación, irán apareciendo pantallas en sentido inverso a como se habían visionado. Android almacena la posición de cada una de las Activities (pantallas) en una “pila”, Stack (también llamada Back Stack), en este caso, de tipo LIFO (Last In, First Out), acrónimo que significa que el último en entrar será el primero en salir.

El sistema de funcionamiento de esta pila es muy simple: cuando la actividad en curso inicia otra, esta nueva actividad obtiene el foco y es empujada a la parte superior de la pila; la actividad anterior permanece en la pila, pero detenida. El conjunto de actividades que tienen una relación lógica y que se encuentran en la pila se denomina tarea (Task).

Si el usuario regresa (Back), la actividad que tenía el foco se elimina, y la actividad previa y que se encontraba por debajo de ella en la pila se restaura y vuelve a coger el foco.

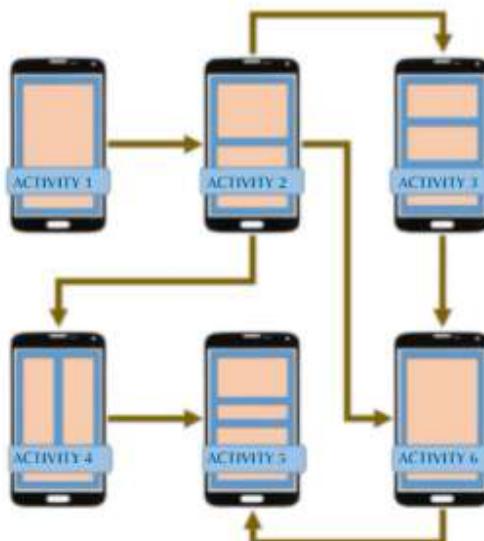


Figura 3.1
Navegación entre actividades de una aplicación.

Si se continúa presionando Atrás, se irán eliminando sucesivamente tareas hasta llegar a la pantalla de inicio. Cuando todas las actividades se han eliminado de la pila, la tarea deja de existir y desaparece de la memoria.

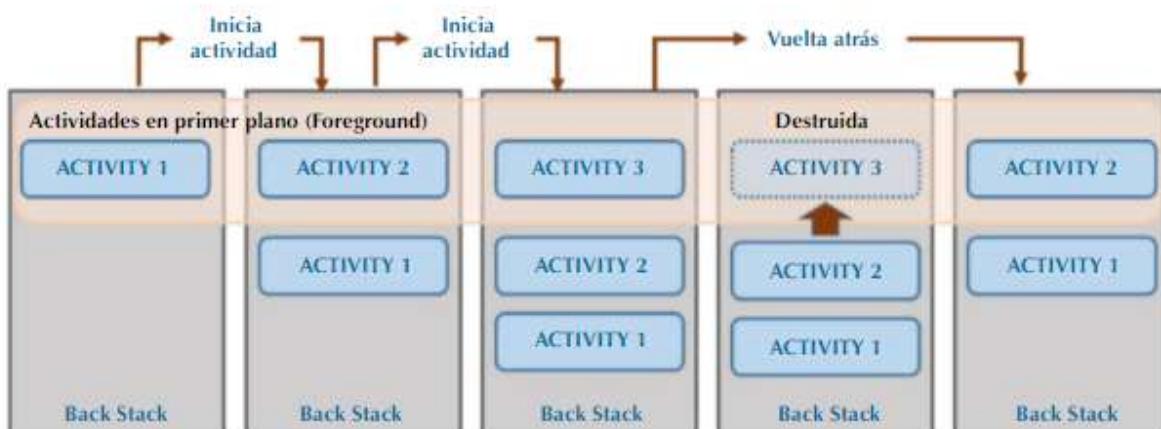


Figura 3.2

Dinámica de funcionamiento del Back Stack.

Una tarea puede moverse en conjunto a un “segundo plano” cuando el usuario comienza una nueva tarea o va a la pantalla Inicio. Si está en segundo plano, todas las actividades de la tarea se paran, pero el Back Stack de esta tarea se conserva. Si una tarea vuelve a situarse en “primer plano”, esta se volvería a posicionar en el lugar donde se había dejado.

RECUERDA

- ✓ Una tarea es un conjunto de actividades con las que el usuario interactúa para realizar un determinado trabajo.

Cuando se detiene una de las actividades, y teniendo en cuenta las características del dispositivo (especialmente la memoria), es posible que el sistema, si necesita recursos de memoria, elimine esa actividad completamente, pero aun así mantiene su lugar en la pila de actividades para cuando necesite volver a primer plano.

WWW

Recurso web

En este código QR encontrarás un vídeo de Android Developer sobre el funcionamiento del Back Stack.



Cuando esto suceda, el sistema debe volver a crearla (en lugar de reanudarla), y para evitar que se pierda el trabajo que tenía en curso, se implementa el método `onSaveInstanceState()`, que asegura la recuperación de variables y otros valores ligados a ella.

Por lo que se ha visto hasta ahora, las actividades funcionan dentro del dispositivo como procesos que pueden estar en diferentes estados o momentos de ejecución.

- Foreground Process.** Es el proceso en que se encuentra la actividad que se está mostrando en pantalla y con la que el usuario puede interactuar.
- Visible Process.** La actividad es visible, pero aún no se puede interactuar con ella. Aunque este estado parezca tener poco sentido, su uso tendrá especial utilidad para el programador, ya que es una situación considerada importante por el sistema operativo.
- Service Process.** Proceso que lleva una actividad que está funcionando como un servicio. Hace cosas en segundo plano y generalmente importantes.
- Background Process.** Aquel que tiene una actividad no visible y situada en un nivel inferior de la pila.
- Empty Process.** Se trata de aquellos procesos que no contienen nada y que Android suele usar para crear sobre ellos un proceso nuevo.

Investiga



Son muchas las posibilidades que te ofrece el conocer de manera correcta la gestión de tareas y de la pila de actividades. Visita la página de Android Developer y familiarízate con su manejo:

3.3. Ciclo de vida de una aplicación

Como se ha visto en el apartado anterior, una actividad puede estar en diferentes períodos de funcionamiento que denominaremos **estados**, y que van a depender del flujo de la aplicación, de la interacción con el usuario y de las necesidades de recursos del sistema.

La actividad debe relacionarse con el sistema operativo y con otros programas que pueden estar en diferentes estados en el dispositivo móvil. Teniendo en cuenta el estado de funcionamiento de la misma, para Android, la actividad puede ser:

- Inexistente.** Cuando no ocupa memoria, por lo que, evidentemente, no se puede interactuar con ella.
- Detenida.** Sí ocupa memoria, aunque no es visible y, por tanto, tampoco es accesible.
- Pausada.** Es visible, pero no se puede interactuar con ella.
- Activa.** Está en primer plano del dispositivo y el usuario puede interrelacionarse con ella.

Estos estados suelen ir en sucesión creciente cuando la actividad se pone en marcha, y en sentido decreciente cuando desaparece y se destruye, por lo que el conjunto de estados en el que se puede encontrar una actividad es comúnmente denominado **ciclo de vida**.

Los cambios de estado pueden ser controlados a través de métodos, llamados por Android tipo **callback**. En la figura 3.3 pueden verse los nombres de dichos métodos en el proceso que transcurre desde la construcción hasta la destrucción de la actividad.

Es necesario aprender a controlar el ciclo de vida de una aplicación y saber manejar cada uno de estos métodos para hacer que la aplicación sea lo más robusta posible.

1. **onCreate()**: es llamado cuando la actividad es invocada por primera vez. Es el momento en que se crean las vistas (el interfaz de usuario), se hace una reserva de memoria para los datos, se obtienen las referencias e instancias de componentes mediante `findViewById()`, se suelen hacer las consultas iniciales en las fuentes de datos y se guardan las variables del entorno, como anteriormente se ha comentado (Bundle).

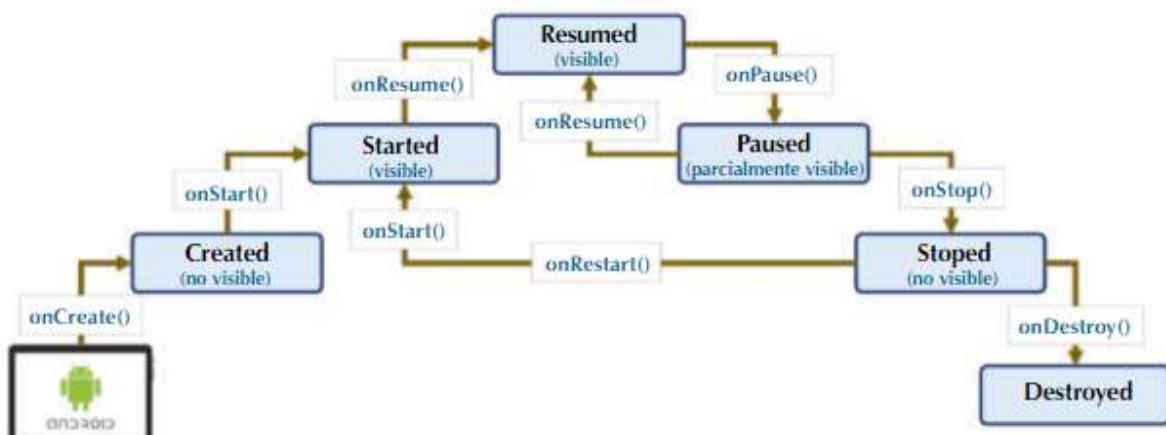


Figura 3.3

Ciclo de vida de una actividad en Android.

2. **onStart()**: este método sucede al anterior. La actividad se hace visible para el usuario, pero aún no se puede interactuar con ella. Suele ser el lugar donde ubicar instrucciones asociadas al interfaz de usuario.
3. **onResume()**: un método que es llamado cuando la actividad puede interactuar con el usuario. Sigue al anterior y precede al `onPause()`. En este periodo se suelen activar sensores, cámara, ejecutar animaciones, actualizar información...
4. **onPause()**: ocurre cuando nuestra actividad pasa a un segundo plano, bien porque está en proceso de destrucción o porque otra actividad pasa a ocupar el primer plano. Es cuando se detienen procesos, animaciones... En este periodo la actividad es parcialmente visible. Puede ocurrir que vuelva a primer plano (`onResume`) o que sea parada `onStop()`, haciéndose invisible para el usuario.
5. **onStop()**: este método es invocado cuando la actividad ya no es visible al usuario y otra actividad ha pasado a primer plano. Se pausan animaciones, determinados servicios se detienen (GPS) y se minimizan los recursos consumidos por la actividad, aunque la actividad aún está en memoria. Si queremos reactivarla, se utiliza `onRestart()`, volviendo a primer plano, o `onDestroy()`, si queremos eliminarla definitivamente.
6. **onRestart()**: llamado cuando una actividad parada vuelve a primer plano. Siempre es seguido de un `onStart()`.
7. **onDestroy()**: es llamado cuando la actividad es definitivamente destruida y eliminada de memoria. En su interior se suelen limpiar y liberar recursos, por ejemplo, la cámara.

Por lo que se ha visto anteriormente, se puede decir que el *ciclo de vida completo* de una actividad transcurre entre la llamada a `onCreate()` y la llamada a `onDestroy()`.

El *ciclo de vida visible* de una actividad es aquel en que el usuario puede ver la actividad en pantalla y que transcurre entre la llamada a `onStart()` y la llamada a `onStop()`.

Por último, el *ciclo de vida en primer plano* de una actividad transcurre entre la llamada a `onResume()` y la llamada a `onPause()`. Es el momento que la actividad tiene el foco en el dispositivo y se encuentra por encima de todas las demás actividades que están activas en memoria.

3.4. ¡Hola mundo de Android!

En este apartado se creará la primera aplicación nativa en Android. Será simple, pero marcará la pauta de cómo se ha de proceder para desarrollar el trabajo práctico propuesto a lo largo de este libro.

Para ello, siguiendo los pasos del capítulo anterior, se ha de tener instalado el entorno de desarrollo con el que trabajaremos (aconsejable Android Studio). Tanto este como el sistema operativo deben reconocer el dispositivo móvil físico o alternativamente tener un emulador configurado con el que visionar las aplicaciones que se irán desarrollando.



Actividad propuesta 3.1

Para crear un nuevo proyecto debes introducir el nombre del proyecto (Proyecto1), el dominio (`android.ejemplo.es`) y la ubicación (es aconsejable que en estos primeros ejemplos utilicen la raíz de alguna de tus unidades). El cuadro de diálogo te muestra cómo será el nombre del paquete de tu aplicación, en este caso, y si has seguido el nombre y dominio propuesto, será `es.ejemplo.android.proyecto1`. A continuación, se deben definir los dispositivos destino, seleccionando el nivel más bajo de API para el que funcionará esta aplicación. Es recomendable seleccionar API 21: Android 5 (Lollipop), con lo que será funcional en un 85% de dispositivos.

En el siguiente paso, se añadirá una Activity a la aplicación y se configurará el aspecto que esta tendrá. Inicialmente, se debe seleccionar una actividad vacía (Empty Activity). Por último, se ha de introducir el nombre de la actividad (Actividad1) y el nombre de su vista (activity_actividad1).

Si se han seguido los pasos anteriores, el entorno habrá generado todos los elementos necesarios de la aplicación, las carpetas estudiadas en el capítulo anterior, incluida Grandle Script, con las librerías y referencias necesarias para la compilación y ejecución de la aplicación.

También estará listo el `AndroidManifest.xml` y los ficheros de código `Actividad1.java` (código de la actividad principal) y dentro de res/layout estará `activity_actividad1.xml` (la vista de la actividad principal).

Antes de estudiar el contenido de cada una de las carpetas, y teniendo conectado un dispositivo móvil al equipo y debidamente configurado, es aconsejable ejecutar la aplicación (pulsando la flecha



Figura 3.4
¡Hola Mundo!

verde o Mayús+F10). Tras lo cual, debe de haber aparecido en pantalla algo similar a la imagen mostrada en la figura 3.4.

En un breve análisis de los dos ficheros que soportan el código de esta primera aplicación, se puede ver en el fichero XML, responsable de la vista de la actividad (ubicado en res/layout/activity_actividad1.xml), que en el panel del mismo Android Studio ofrece dos posibilidades de trabajo, vista diseño (Design) y modo texto. Es recomendable, como desarrollador, acostumbrarse a esta segunda, construyendo la vista mediante código, aunque en algunos casos parezca más cómodo la vista gráfica.

El código contenido en la vista se estudiará más adelante, pero ahora basta con conocer que consta de un contenedor (en este caso, un ConstraintLayout) y un objeto de texto en su interior (TextView); este texto es el que ha aparecido en pantalla. Además, también vienen definidos la codificación y el esquema de Android que le permite reconocer los comandos básicos.

Nos detendremos un poco más en el estudio del fichero Proyecto1.java (se encuentra dentro de la carpeta java). Si se han utilizado los parámetros propuestos, esto es lo que debe aparecer:

```
package es.ejemplo.android.proyecto1;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class Actividad1 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad1);
    }
}
```

Vamos a analizar este código, ya que seguirá un modelo de uso común en la programación en Android. Lo primero que aparece es el paquete en el que irá envuelta la aplicación que, en este caso, es una simple clase de Java, pero con la peculiaridad de que hereda (extends) de la superclase AppCompatActivity, lo que proporciona a esta clase la posibilidad de “entender” el código específico de Android.

Esta es la superclase que se suele utilizar desde la API 15, aunque, como se verá más adelante, en algunas ocasiones se necesitará hacer que la clase herede de otras superclases.



PARA SABER MÁS

La gran superclase en la que se basa Android es Activity. Esta es la clase base y todas las actividades deben heredar de ella, directa o indirectamente. La evolución posterior de las diferentes versiones de Android ha hecho que esta clase haya ido relegando funciones en otras superclases. Siendo la relación entre ellas la siguiente y significando en este caso el símbolo < que hereda de...

Activity < FragmentActivity < AppCompatActivity < ActionBarActivity

Por tanto, se puede entender que `android.support.v7.app.AppCompatActivity`, uno de los import que precede a la clase, facilita a la aplicación las librerías que contiene esta superclase.



Actividad propuesta 3.2

Elimina la extensión de la superclase y observa el efecto que tiene sobre el código. Verás que deja de reconocer los métodos ligados al ciclo de vida de una actividad de Android. Y si eliminas o comentas el import de la superclase, además de no reconocer los métodos, tu aplicación tampoco reconocerá la superclase.

Dentro de la clase Actividad1, tan solo se incluye un método, sobrescrito (@Override), que recordando el apartado anterior, responderá en el momento que la aplicación se cree y entre en funcionamiento en la memoria del teléfono.

Los otros momentos del ciclo de vida de la aplicación sucederán a este, aunque no aparece ninguno de ellos, al no haberse escrito código para los métodos que responden a cada uno de los demás estados.

En el momento que se crea la aplicación, ocurren dos hechos importantes. Por un lado, se prepara un Bundle (paquete de almacenaje) donde guardar las instancias y estados de la aplicación para ser recuperados en caso de necesidad. Para ello también existe una librería (import) que suministra este paquete.

Evidentemente, en este ejemplo hay muy poco que guardar, pero es fácil imaginar lo que puede ocurrir si no existiera esta posibilidad y en pleno funcionamiento de una aplicación, especialmente si tiene formularios de datos, entra una llamada de teléfono y la aplicación se pone en pausa.

Preparado el almacenaje, se invoca `savedInstanceState`, la llamada a la superclase que se ocupa de gestionar el salvado. Más adelante, se profundizará en el guardado de estados en la actividad.

El otro hecho importante que ocurre, cuando se crea la actividad, viene marcado con la línea de código `setContent View(R.layout.activity_actividad1)`. Esta lo que hace es asociar el fichero de la vista a la clase Java, utilizando para ello la clase R como gestora. Y tan solo con eso se “inflará” el contenido visual definido dentro del fichero XML antes visto.

Estos dos puntos son principios muy elementales, pero imprescindibles para dar los primeros pasos en el conocimiento de Android.

Si se añade el siguiente código a la aplicación, se consigue un Log informativo que avisa del paso por el momento del ciclo de vida Startet (para que reconozca este comando, Android Studio importará la librería `android.util.Log`).

Es recomendable filtrar, en el entorno de desarrollo, los Log por tipo (Verbose, Debug, Info, Warn, Error, Assert), además de establecer un filtro personalizado por Tag (en este caso, el Tag definido es “EJEMPLO”).

```
@Override
protected void onStart(){
    super.onStart();
    Log.i("EJEMPLO", "Estoy onStart");
}
```

Actividad propuesta 3.3



Siguiendo el ejemplo anterior, implementa Logs en tu aplicación que informen del paso por cada uno de los estados posibles del ciclo de vida de la misma. El resultado debe ser algo similar a la siguiente captura de pantalla. No olvides que una aplicación pasa al estado de parado cuando pasa a segundo plano.

```

2019-08-04 21:32:52.572 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onCreate
2019-08-04 21:32:52.877 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onRestart
2019-08-04 21:32:52.592 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onPause
2019-08-04 21:32:55.197 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onpause
2019-08-04 21:32:55.542 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onStop
2019-08-04 21:33:02.990 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onstop
2019-08-04 21:33:03.001 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onstart
2019-08-04 21:33:03.002 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onResume
2019-08-04 21:33:04.590 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onresume
2019-08-04 21:33:04.931 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onstop
2019-08-04 21:33:04.932 24933-24933/es.ejemplo.android.project1 I/EJEMPLO: Estoy onDestroy

```

3.5. Conservar el estado de una aplicación

Como ya se ha visto en el ciclo de vida, cuando se pausa o se detiene una actividad, se conserva el estado de la misma. Esto ocurre porque el objeto Activity aún se conserva en memoria, y toda la información de sus objetos y su estado actual permanece. Por tanto, cuando la actividad regrese a primer plano, esa información aún estará ahí.

Puede ocurrir que, estando parada, el sistema destruya una actividad para recuperar recursos de memoria; el objeto Activity también se destruye, lo que es desconocido para el usuario y, si este quiere volver a pasarla a primer plano, el sistema debe volver a crear el objeto Activity, para aparecer con el mismo aspecto e información que el usuario la dejó. Todo esto se consigue mediante la implementación de un método de callback, que permite guardar información acerca del estado de nuestra actividad: `onSaveInstanceState()`.



Figura 3.5
Gestión de memoria y salvado de instancias.

Antes de la destrucción, el sistema llama a `onSaveInstanceState()`, método que es pasado a un Bundle, donde puede ser guardada la información del estado, mediante métodos como `putString()` y `putInt()`. Con la aplicación destruida, cuando el sistema vuelve a crear la actividad se extrae el estado guardado en el Bundle; si no hay información que recuperar (como ocurre cuando se crea por primera vez), el Bundle que se recibe es null.

TOMA NOTA



La llamada a `onSaveInstanceState()` va ligada de manera específica a cada objeto de la actividad (especialmente de la vista), que conserva el estado de cada uno de ellos, proporcionando en cada vista información sobre sí misma en el momento de la restauración. Para que esto ocurra, es necesario que se proporcione un identificador a cada uno de los objetos que aparezcan en la aplicación (con el atributo `android:id`), de tal manera que si un objeto no tiene ID, el sistema no puede guardar su estado.

Puede ocurrir que cuando el usuario provoque la destrucción de la actividad (saliendo de ella al pulsar el botón de atrás), no se produzca llamada a `onSaveInstanceState()` y, por tanto, no se conserve la información existente. El salvado de estados puede ser deshabilitado, lo que no es aconsejable, mediante el atributo `android:saveEnabled` en `false` o llamando al método `setSaveEnabled()`.

3.6. Intents y filtros

Un Intent es un sistema de comunicación que permite interactuar entre componentes de la misma o de distintas aplicaciones de Android. Es un elemento básico de comunicación, tal que con él se puede comenzar una aplicación, iniciar un servicio, entregar un mensaje.

Mediante un Intent se podrá enviar o recibir información desde y hacia otras actividades o servicios, así como lanzar mensajes de tipo Broadcast para identificar cuándo han ocurrido ciertos eventos. Es una manera útil de reutilizar otros componentes que ya desarrollan la funcionalidad que necesitamos en nuestro código, pudiendo, por ejemplo, iniciar una aplicación que nos permita tomar fotografías sin tener que realizar el código de dicha aplicación.

Por sus características y construcción, Android dispone de dos tipos de Intents, explícitos o implícitos.

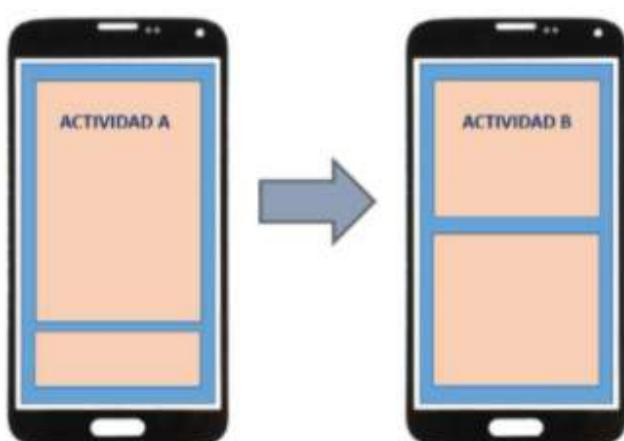


Figura 3.6
Llamada mediante Intent.

3.6.1. Intents explícitos

Los Intents explícitos son aquellos que nombran el componente que se necesita ejecutar, la clase Java específica que se necesita para realizar alguna tarea.

Su construcción es fácil, ya que tan solo se debe instanciar el Intent pasándole como parámetro el contexto y la actividad que se va a lanzar. Posteriormente, lanzarlo mediante el comando `startActivity`.

TEN EN CUENTA

- ✓ Todas las actividades que se ejecuten de manera independiente deben de ser previamente declaradas en el Manifest.

Actividad propuesta 3.4



En la aplicación creada anteriormente con el nombre de Proyecto1, crea una segunda actividad denominada Actividad2 (te recomiendo que, por rapidez, copies, pegues y renombres Actividad1). Igualmente debes hacer con el fichero XML de la vista (renómbralo con `activity_actividad2.xml`). Ahora asocia en la segunda actividad la vista a este fichero. El código de esta debe quedarte así:

```
public class Actividad2 extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad2);
    }
}
```

Retoca el objeto de texto del segundo XML y sustituye el "Hello World!" por otra cadena distinta. A continuación, añade el siguiente código al método `onDestroy` de la clase Actividad1.

```
Intent ejemplo = new Intent(this, Actividad2.class);
startActivity(ejemplo);
```

Si ahora ejecutas y pulsas el botón Atrás del teléfono, verás que se destruye tu aplicación y da error, esto es debido a que no has declarado esta segunda actividad en el Manifest. Debes hacerlo escribiendo el siguiente código después de la etiqueta de cierre (`</activity>`) de la primera actividad, dentro del fichero manifest.

```
<activity android:name=".Actividad2">
</activity>
```

Si ahora ejecutas y pulsas el botón de Atrás del dispositivo verás que se cierra la Actividad1 y, posteriormente, se abre la Actividad2.

3.6.2. Intents implícitos

Los implícitos son aquellos con los que tan solo informamos al sistema la acción que deseamos realizar, y el sistema nos responde con el componente más adecuado para realizar dicha petición, es decir, no se especifica un componente concreto. De tal manera, que si se quisiera ver una página web, sería el sistema el que propondría el navegador con el que hacerlo o preguntaría con cuál hacerlo si tuvieras más de uno instalado en el dispositivo y ninguno configurado como predefinido.



Actividad propuesta 3.5

En la aplicación creada anteriormente con el nombre de Proyecto1, añade el siguiente código al método `onDestroy` de la clase Actividad2. Verás que cuando eliminas esta se te abre el navegador con la página de Google (para cargar las librerías tan solo debes posicionarte sobre los elementos desconocidos y pulsar Alt+Intro).

```
Intent ejemplo= new Intent(Intent.ACTION_VIEW);
ejemplo.setData(Uri.parse("http://www.google.es"));
startActivity(ejemplo);
```

3.6.3. Elementos

Si se observa el código anterior a la hora de construir el Intent, se define la acción que se va a realizar y se aportan los datos con los que realizará dicha acción. Además de estos, los elementos que pueden necesitarse para construir un Intent implícito son los siguientes:

A) Nombre del componente

Utilizado fundamentalmente en los explícitos. Es donde se identifica el componente que se desea lanzar con el Intent.



SABÍAS QUE...

Puedes nombrar un componente de varias maneras:

Usando `setComponent()`

```
ComponentName componente = new ComponentName(this, Actividad.class);
intent.setComponent(componente);
```

Usando `setClass()`

```
intent.setClass(this, Actividad.class);
```

Usando `setClassName()`

```
intent.setClassName("es.ruta", "es.ruta.Actividad");
startActivity(intent);
```

B) Acción

Mediante `setAction()` se suministra una cadena de texto que informa de la acción que se va a realizar y lo identifica dentro del Manifest. La acción determina, en gran parte, los otros elementos del Intent (datos y extras). Existe una serie de acciones bastante comunes y de frecuente uso, como son:

- ACTION_CALL
- ACTION_SET_WALLPAPER
- ACTION_DELETE
- ACTION_VIEW
- ACTION_SEND

En este ejemplo, se invoca al sistema para realizar una llamada telefónica:

```
Intent ejemplo= new Intent(Intent.ACTION_CALL);
ejemplo.setData(Uri.parse("tel:NÚMERO"));
activity.startActivity(ejemplo);
```

Este otro ejemplo abre la cámara del dispositivo:

```
Intent ejemplo= new Intent(android.media.action.IMAGE_CAPTURE);
activity.startActivity(ejemplo);
```

Las acciones pueden clasificarse en varios tipos:

- Acciones genéricas
- Acciones auxiliares
- Acciones definidas por el usuario

C) Datos

Contiene la URI (Uniform Resource Identifier) con los datos con los que el componente que reciba el Intent debe trabajar. El tipo de datos determina qué componente usará el sistema para procesarlos. Así, por ejemplo, para construir un Intent implícito que le pida al sistema enviar un correo electrónico, debería hacerse de la siguiente forma.

```
Intent correo= new Intent();
correo.setAction(Intent.ACTION_SEND)
correo.putExtra(Intent.EXTRA_TEXT, mensaje);
correo.setType("text/plain");
```

D) Categoría

De uso en los implícitos, complementa a la acción, suministrando información adicional sobre el tipo de componente que ha de ser lanzado.

- CATEGORY_BROWSABLE
- CATEGORY_HOME
- CATEGORY_LAUNCHER
- CATEGORY_PREFERENCE

E) Extras

Información adicional que será recibida por el componente lanzado. Muy útil en los explícitos, ya que te servirán para mandar información de una Activity a otra. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase Bundle. Para llenar el Bundle con información, puede utilizarse alguna de estas posibilidades.

```
putExtra(String name, boolean value);
putExtra(String name, int value);
putExtra(String name, double value);
putExtra(String name, String value);
```

Para recoger la información en destino se utiliza:

```
Bundle b = intent.getExtras();
```

F) Flags

Son metadatos que indican al sistema Android cómo iniciar una actividad y determinar, en parte, el comportamiento del componente que atienda la petición.

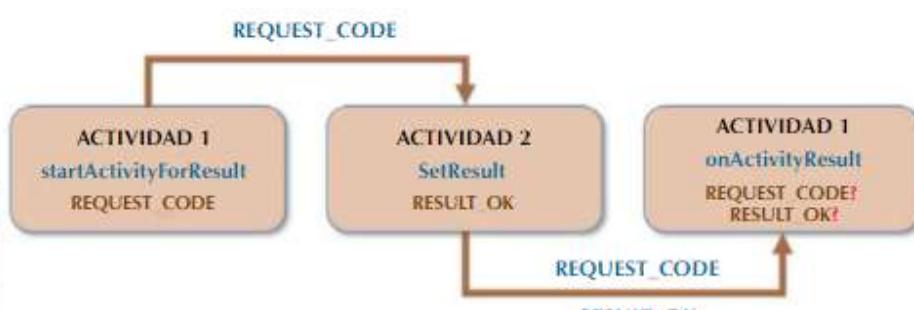
3.6.4. Propagación

Conocidos los tipos (explícito e implícito), elementos y forma que tienen de construirse los Intents, pueden dársele tres usos fundamentales.

A) Para comenzar una actividad

En este caso, además de iniciar la actividad mediante la forma vista, `startActivity()`, puede recibirse el resultado de la actividad cuando esta finalice, mediante `startActivityForResult()`. El proceso se hace utilizando el REQUEST_CODE (información recibida) y el RESULT_OK que confirma que la Activity ha finalizado correctamente.

Figura 3.7
Esquema
de funcionamiento
de `startActivity
ForResult`.



Para ponerlo en práctica, se puede utilizar el siguiente código en la Actividad 1:

```
private static final int REQUEST_CODE = 10;
-----
Intent ejemplo = new Intent(this, ActividadDos.class);
activity.startActivityForResult(ejemplo, REQUEST_CODE);
```

En el método finish() de la Actividad 2:

```
@Override
public void finish() {
    Intent data = new Intent();
    data.putExtra("RETORNO", "Este es el valor de retorno");
    setResult(RESULT_OK, data);
    super.finish();
}
```

Por último, y de nuevo en la Actividad 1, se comprobará que la información que le llega tiene los códigos correctos para que reciba la información.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
        if (data.getStringExtra("RETORNO")) {
            //Código para realizar
        }
    }
}
```

www

Recurso web

En este código QR encontrarás un vídeo tutorial de programación y más de cómo construir una actividad haciendo uso del método onActivityResult.



B) Para iniciar un servicio

Un Service es un componente que realiza operaciones en segundo plano, sin interfaz de usuario. Mediante un Intent se puede iniciar un servicio, modificar su comportamiento o establecer conexión entre componentes. El Intent describe el servicio que se debe iniciar y contiene los datos necesarios para ello. Se pueden utilizar los comandos startService(), stopService() y bindService(). Este último caso sirve para establecer un enlace con el servicio de otro componente mediante un Intent.

C) Para entregar un mensaje

Un mensaje es un aviso que cualquier aplicación registrada puede recibir. Cuando se crea un nuevo receptor Broadcast, se informa al sistema acerca de qué tipos de Intents implícitos pueden ser lanzados con este componente. Pueden enviarse mensajes a otras aplicaciones mediante sendBroadcast(), sendOrderedBroadcast(), sendStickyBroadcast() o sendStickyOrdererBroadcast().

Investiga

Busca en la página de Android Developer información del uso de cada una de las opciones de los Intents en los casos de Service y Broadcast.

3.6.5. Filtros

De utilidad en el Manifest, determinan la acción que puede llevar a cabo un componente y los tipos de Intents que pueden gestionarla. Los filtros se pueden clasificar por acción, por categoría o por tipo de datos.

3.6.6. PendingIntent

Es un tipo de Intent especial en el que se especifica una acción que se va a realizar en el futuro. Pasa un Intent futuro a otra aplicación, así como los permisos necesarios, como si la aplicación creadora estuviese aún en vigor. Puede ser ejecutado en un determinado momento programado mediante el AlarmManager o provocar que se ejecute cuando el usuario realice alguna acción. Suelen ser frecuentemente usados con las notificaciones. Ofrecen tres posibilidades:

- PendingIntent.getActivity(): recupera un PendingIntent para comenzar una actividad.
- PendingIntent.getBroadcast(): Recupera un PendingIntent para escuchar una emisión.
- PendingIntent.getService(): recupera un PendingIntent para iniciar un servicio.
- PendingIntent.getForegroundService(): esto iniciará un ForegroundService.

En el siguiente ejemplo se construirá una actividad que lanzará mediante PendingIntent a una segunda actividad después de transcurrir cinco segundos. Para ello, primero se construye un Intent explícito, que lanzará una actividad denominada Actividad3 (se puede utilizar cualquier actividad creada para ello). Luego se le asocia un PendingIntent, al que se le ha de pasar como parámetros el contexto el requestCode (necesario en caso de cancelarlo), el Intent y el Flag.

```
Intent intent = new Intent(this, Actividad3.class);
PendingIntent miPending = PendingIntent.getActivity(
    this, 1, intent, miPending.FLAG_UPDATE_CURRENT);
```



PARA SABER MÁS

En este caso, se ha utilizado un Flag de tipo FLAG_UPDATE_CURRENT (actualiza este PendingIntent en caso de existir). Otras opciones son:

- FLAG_CANCEL_CURRENT: si existe el PendingIntent debe cancelarse.
- FLAG_ONE_SHOT: el PendingIntent solo puede usarse una vez.
- FLAG_IMMUTABLE: el PendingIntent no puede cambiar.

El siguiente paso será ligar el PendingIntent al AlarmManager para que responda después del tiempo indicado. Se instancia este servicio y se configura según el tiempo deseado (no hay que olvidar que el tiempo Android lo trata en milisegundos). El código resultante sería el siguiente:

```
AlarmManager alarma= (AlarmManager) getSystemService(ALARM_SERVICE);
alarma.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + (5 * 1000), miPending);
```

Si ahora se ejecuta la aplicación, pasados cinco segundos se inicia la aplicación que será utilizada en lugar de Ejemplo.class. Esto ocurrirá de igual manera si se cierra antes de los cinco segundos la aplicación Padre que gestiona el Intent.

Resumen

- Una actividad (Activity) es el principal componente de las aplicaciones en Android, equivale a una pantalla y se ocupa de las interacciones con el usuario. En caso de tener más de una actividad en memoria, el sistema se ocupa de almacenarlas en una pila (Back Stack), que es de tipo LIFO. El conjunto de actividades que tienen una relación entre ellas se denomina tarea (Task).
- Las actividades funcionan dentro de nuestro dispositivo como procesos que pueden estar en diferentes estados: Foreground, Visible, Service, Background y Empty Process.
- El estado en que se encuentra en cada momento una actividad forma parte del denominado **ciclo de vida**. Los cambios de estado pueden ser controlados a través de los siguientes métodos: onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart() y onDestroy().
- Según la visibilidad y la interacción con el usuario, los estados de una actividad pueden ser incluidos en un ciclo de vida completo (todos los estados), ciclo de vida visible (puede verse la actividad) y ciclo de vida en primer plano (puede interactuarse con ella).
- En caso de necesidad de recursos, el sistema puede provocar la destrucción temporal de una actividad que el usuario tenía parada. Cuando esta vuelve a primer plano, el sistema debe volver a crearla, recuperando íntegramente el estado anterior gracias al método onSaveInstanceState().

- Los Intents (intenciones) son componentes básicos de comunicación que permiten interactuar entre componentes de la misma o de distintas aplicaciones de Android. Pueden ser utilizados para comenzar una actividad, iniciar un servicio o entregar un mensaje.
- Existen Intents de tipo explícito (cuando se nombra el componente a ejecutar) o implícito (cuando se dice al sistema lo que se desea hacer y él decide el componente más apropiado para realizar dicha acción). Tanto uno como otro pueden estar constituidos por una serie de elementos, como son: nombre, acción, datos, categoría, extras y flags. También se le pueden aplicar filtros para determinar la acción que se va a realizar.
- Un tipo especial de Intent es el PendingIntent, aquel en el que se especifica una acción que se va a realizar en el futuro, ejecutándose en un momento programado como si la aplicación creadora estuviese aún en vigor.

Actividades de Autoevaluación

1. ¿Cómo se denomina el proceso cuando el usuario puede interactuar con él?:

- a) Foreground Process.
- b) Visible Process.
- c) Service Process.
- d) Background Process.

2. Cuando una actividad es visible y se puede interactuar con ella, ¿en qué estado se encuentra?:

- a) Created.
- b) Started.
- c) Resumed.
- d) Paused.

3. ¿Qué método suele utilizarse para incluir la creación de las vistas?:

- a) onCreate().
- b) onStart().
- c) onResume().
- d) onDestroy().

4. ¿Cuál de estos métodos no se incluye en el ciclo de vida visible?:

- a) onCreate().
- b) onStart().
- c) onResume().
- d) onPause().

5. ¿Dónde realiza el salvado de instancias el método savedInstanceState?:
- a) En un recurso XML.
 - b) En un Bundle en memoria.
 - c) En un fichero SQLite.
 - d) En un fichero de texto plano.
6. ¿Cómo se denominan los Intents que son llamados por la acción que se va a realizar?:
- a) Explícitos.
 - b) Implícitos.
 - c) Diferidos.
 - d) PendingIntent.
7. ¿Cuál de los siguientes no es incluido entre los elementos de un Intent?:
- a) Nombre del componente.
 - b) Categoría.
 - c) Package.
 - d) Flags.
8. ¿Cuál de los siguientes no es un uso frecuente de un Intent?:
- a) Comenzar una actividad.
 - b) Iniciar un servicio.
 - c) Almacenar información.
 - d) Entregar un mensaje.
9. ¿Cuál de los siguientes métodos es incorrecto para entregar un mensaje mediante Intent?:
- a) sendBroadcast().
 - b) sendOrderedBroadcast().
 - c) sendStickyBroadcast().
 - d) sendOrdererStickyBroadcast().
10. Con un PendingIntent:
- a) Podemos realizar acciones diferidas.
 - b) Podemos construir notificaciones.
 - c) Podemos iniciar aplicaciones a la espera de resultados.
 - d) Podemos realizar acciones sin necesidad de que la actividad creadora esté activa.

SOLUCIONES:

1. **a** b c d
 2. a b **c** d
 3. **a** b c d
 4. **a** b c d

5. a **b** c d
 6. **a** b c d
 7. a b **c** d
 8. a b c d

9. a b c **d**
 10. a b c d

Interface de usuario: los layouts

Objetivos

- ✓ Identificar la diferencia entre la clase View y la ViewGroup.
- ✓ Familiarizarse con los atributos comunes a estas clases.
- ✓ Manejar con soltura los atributos de posicionamiento.
- ✓ Saber aplicar cada tipo de layout según las necesidades de la aplicación.
- ✓ Trabajar sin dificultad con el anidamiento de layout.
- ✓ Descubrir la variedad de uso del TableLayout y del GridLayout.
- ✓ Posicionar sin dificultad los objetos en un RelativeLayout.
- ✓ Conocer la relación entre diseño y código en los ConstraintLayouts.

Mapa conceptual



Glosario

Atributo. Un atributo define la propiedad de un objeto, elemento o archivo. También puede establecer el valor para una instancia determinada de los mismos.

Interfaz. Conexión funcional entre dos componentes, dispositivos o sistemas, facilitando la comunicación entre ellos y permitiendo el intercambio de información.

match_parent. Es un parámetro que fuerza al objeto a expandirse para ocupar todo el espacio disponible en el elemento de diseño que lo contiene.

wrap_content. Establece el tamaño de un contenedor forzándolo a expandirse solo lo suficiente como para contener los objetos que almacena.

4.1. Introducción

Hasta ahora se han visto los fundamentos y componentes de una aplicación en Android, también su ciclo de vida y los elementos que forman la estructura de un proyecto, además de conocer cómo se estructuran los recursos que pueden ser necesarios a la hora de desarrollar

una aplicación. También se han dado los primeros pasos en programación creando una pequeña aplicación, con una actividad simple e incluso haciendo que esta llame a una segunda actividad.

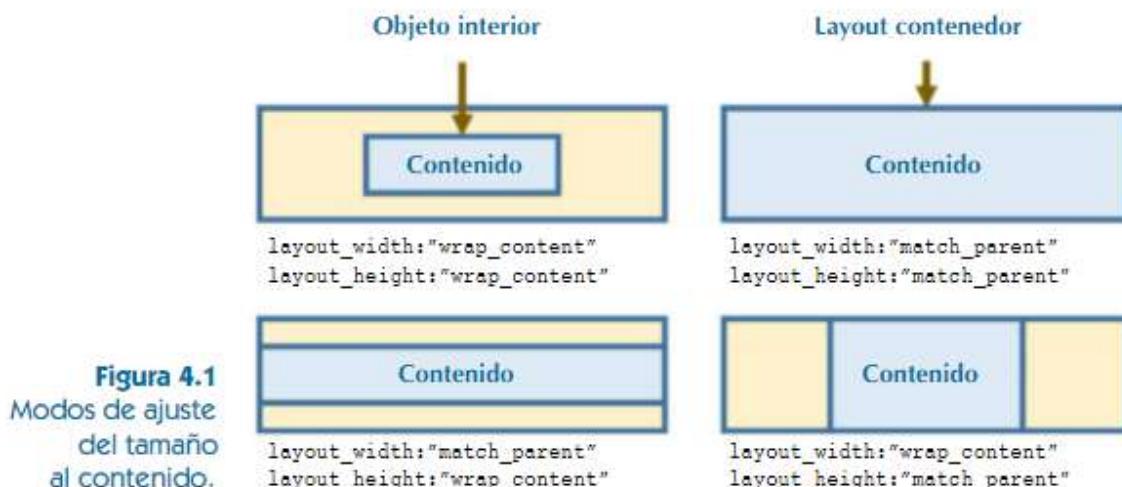
A partir de este capítulo se irán siguiendo pasos de complejidad creciente hasta adquirir soltura en el desarrollo de aplicaciones nativas. Es sumamente necesario que se enfoque cada uno de los apartados desde un punto de vista práctico, realizando las actividades propuestas, tanto de ejercicios de seguimiento como de ampliación.

Se iniciará este aprendizaje trabajando los elementos de interfaz, abordando primeramente para ello los layouts como objetos contenedores de las vistas y de necesario conocimiento para progresar en los siguientes capítulos de este libro.

4.2. Interfaces de usuario. Clases asociadas

El interfaz de una aplicación Android es aquello que aparece en pantalla, que el usuario puede ver y, por tanto, interactuar con él. Android nos ofrece un gran número de componentes implementados y preparados para su uso en la interfaz, en los cuales la clase View sirve siempre como clase base.

La librería android.view nos proporciona los elementos de la interfaz para construir las vistas, lo que se puede hacer usando Java, en principio más complejo y, por tanto, menos eficiente, o bien definiéndolo en el archivo XML ubicado en el directorio res/layout.



La clase View como clase principal en la jerarquía de vistas tiene una serie de atributos que serán heredados por todos los elementos que de ella provienen. A cualquiera de ellos le puedes asignar valores medidos (entre comillas) en pulgadas (*in*), milímetros (*mm*), puntos (*pt*), píxeles (*px*), píxeles independientes de la densidad (*dp*) y píxeles independientes de la escala (*sp*). Más adelante (apartado pantalla) se verá el significado y utilidad de cada uno; por el momento, es recomendable utilizar como unidad de medida los *píxeles independientes de la densidad (dp)*.

CUADRO 4.1**Atributos**

Atributos de posicionamiento	layout_width	ancho
	layout_height	alto
Atributos para los márgenes	layout_margin	cuatro márgenes
	layout_marginBottom	margen inferior
	layout_marginLeft	margen izquierdo
	layout_marginRight	margen derecho
	layout_marginTop	margen superior
Atributos para el espaciado	android:padding	espaciado a los cuatro lados
	android:paddingTop	espaciado superior
	android:paddingBottom	espaciado inferior
	android:paddingLeft	espaciado izquierdo
	android:paddingRight	espaciado derecho

**TOMA NOTA**

Para posicionar los objetos que heredan de la clase View además de poder tomar valores absolutos (en cualquiera de las unidades citadas), también pueden asignársele los valores relativos *wrap_content*, cuando se desea ajustar el tamaño al contenido del objeto o *match_parent* (antes *fill_parent*) para tomar el máximo tamaño que le permite el contenedor.

Si se desea centrar o justificar la vista, se utilizará el atributo *layout_gravity*, y para distribuir el espacio disponible entre los diferentes objetos, *layout_weight*. Por último, para poder identificar el objeto debe utilizarse *@id+/nombre* y para acceder a este identificador, *@id/nombre*. También pueden utilizarse identificadores definidos por el sistema *@android:id/nombre*.

Dentro de esta gran clase se encuentra la clase android.view.ViewGroup. Ella proporciona, como su propio nombre indica, los objetos ViewGroup, cuya función es contener y controlar colecciones de View o de otros ViewGroup.

Los objetos Views han de ser hijos de objetos ViewGroup, por lo tanto, se ha de comenzar el diseño de la interfaz con un objeto ViewGroup, a

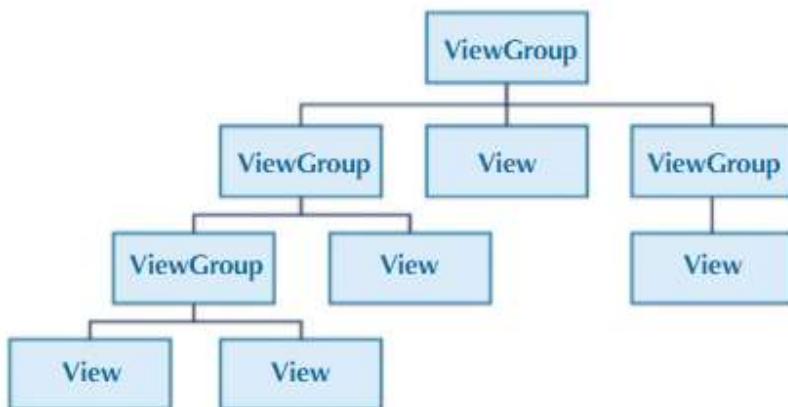


Figura 4.2
Jerarquía de la clase View.

partir del cual se puede realizar un árbol con tantas bifurcaciones como se desee. Utilizaremos la clase ViewGroup como base de estudio de la clase layouts, que son subclases que proveen de los tipos más comunes de layouts de pantalla.

RECUERDA

- ✓ Los atributos de posicionamiento deberán usarse siempre que se coloque un objeto en la vista de la aplicación.
- ✓ Los atributos de espaciado y márgenes tan solo se usarán cuando se deseé mejorar la estética de la aplicación.
- ✓ Siempre es recomendable identificar los objetos en la vista aunque luego no se vaya a acceder a ellos desde el Java de la aplicación.

4.3. Layouts

Los layouts son elementos no visuales destinados a controlar la distribución, la posición y las dimensiones de los controles que se insertan en su interior. Dentro de cada uno pueden ubicarse todos los elementos que sean necesarios en el interfaz de la actividad, incluidos otros layouts, lo que permitirá estructurar la pantalla de la manera deseada. En función de la forma y el posicionamiento en pantalla, existe gran variedad de layouts, que se estudiarán a continuación.

4.4. LinearLayout

Este layout apila uno tras otro todos sus elementos hijos de forma horizontal o vertical, según se establezca su atributo android:orientation.

Si se escribe el siguiente código, tan solo aparecerá una pantalla en rojo.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#FF0000">
</LinearLayout>
```

A continuación, se anidarán, en vertical, tres layout; el central será de triple tamaño que los otros dos y se pondrán en azul los dos más pequeños.

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

```

    android:orientation="vertical">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="#0000ff">
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:background="#fffff">
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:background="#0000ff">
    </LinearLayout>
</LinearLayout>

```

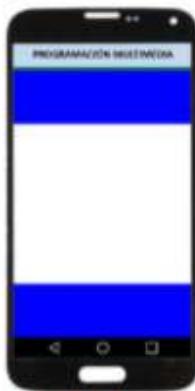


Figura 4.3
Layouts
en vertical.

Cuando una aplicación tiene un contenido de diseño, en el interfaz, con mayor tamaño que la altura del dispositivo y se necesita hacer el contenido desplazable, se utilizará ScrollView para desplazamiento vertical o HorizontalScrollView para desplazamiento horizontal.

RECUERDA

- ✓ Con `android:background` se puede colorear el fondo de los objetos de la vista.
- ✓ `layout_weight` es un atributo que permite dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellos.
- ✓ El código `xmlns:android="http://schemas.android.com/apk/res/android"` deberá usarse en el layout principal de la aplicación y sirve para declarar el espacio de nombres donde esta trabajará.

4.5. FrameLayout

Coloca todos sus controles hijos alineados con su esquina superior izquierda, sin distribuirlos, de forma que cada control quedará oculto por el control siguiente.

Este contenedor ofrece la posibilidad de modificar la visibilidad del elemento deseado dentro de su contenido, para lo cual será necesario utilizar la propiedad `android:visibility`, lo que es muchas veces útil para determinados efectos de animación. Para colocar los objetos hijos en la posición deseada, podemos utilizar la propiedad `android:gravity`.

TEN EN CUENTA

- ✓ No debe confundirse `android:gravity` con `android:layout_gravity`: el primero establece la posición del contenido y el segundo posiciona el objeto con respecto a su elemento padre.

```

<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/frameLayout" >
    <FrameLayout
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#E00000"
        android:layout_gravity="left|bottom">
    </FrameLayout >
    <FrameLayout
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#00ff00"
        android:layout_gravity="right|top">
    </FrameLayout >
</FrameLayout>

```

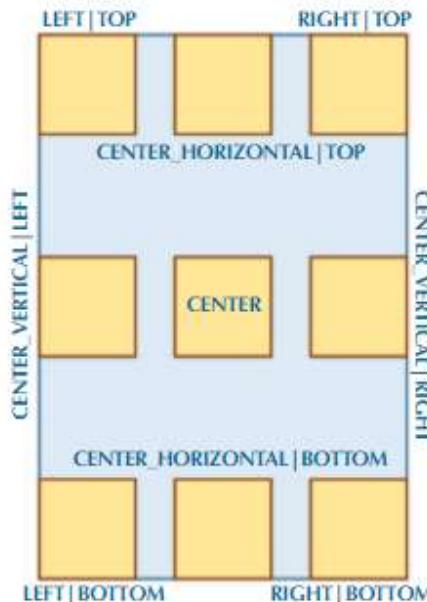


Figura 4.4
Ubicación en FrameLayouts.



Actividades propuestas

- 4.1.** Realiza con LinearLayout la siguiente estructura de layouts anidados.



- 4.2.** Realiza una aplicación con una actividad cuyo FrameLayout principal tenga nueve FrameLayouts interiores y muestren la disposición de la figura 4.4.

4.6. AbsoluteLayout

Permite indicar las coordenadas absolutas de pantalla (x-y) donde se quiere visualizar cada elemento:

`android:layout_x="posición horizontal"`

`android:layout_y="posición vertical"`

Los fabricantes crean dispositivos de muy variada resolución, por lo que Android nos permite un rango de resoluciones de pantalla muy variado, lo que hace que el contenedor distribuya los elementos que acoge según sus propias reglas, y que el resultado pueda no ser el deseado. Por ello, no es recomendable utilizar este tipo de contenedor, que hoy día ha sido marcado como obsoleto.

4.7. RelativeLayout

En este layout los elementos van colocados en una posición relativa unos a otros, especificando la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. Las opciones de posicionamiento disponibles son las siguientes:

```

Posición relativa al control
- android:layout_above
- android:layout_below
- android:layout_toLeftOf
- android:layout_toRightOf

Alineación con respecto al control
- android:layout_alignLeft
- android:layout_alignRight
- android:layout_alignTop
- android:layout_alignBottom
- android:layout_alignBaseline

Posición relativa al layout padre
- android:layout_alignParentLeft
- android:layout_alignParentRight
- android:layout_alignParentTop
- android:layout_alignParentBottom
- android:layout_centerHorizontal
- android:layout_centerVertical
- android:layout_centerI

```

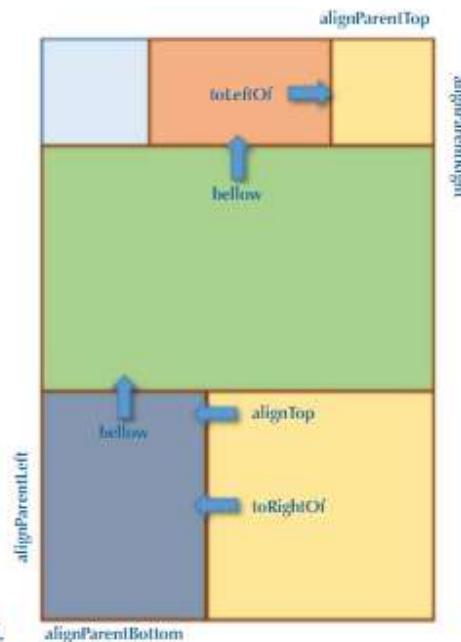


Figura 4.5
Relaciones en un
RelativeLayout.

Actividad propuesta 4.3



El código para posicionar los dos primeros layouts (fila superior) de la figura 4.5 es el que se expone a continuación. Realiza un proyecto cuya vista, además de estos, tenga todos los mostrados en dicha figura.

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/padre">
    <RelativeLayout
        android:id="@+id/hijo1"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:background="#ff0000">
    </RelativeLayout>
    <RelativeLayout
        android:id="@+id/hijo2"
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@+id/hijo1"
        android:background="#0000ff">
    </RelativeLayout>
</RelativeLayout>

```

4.8. TableLayout

Este layout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla. Realmente, esta modalidad es un tipo especial de LinearLayout, con una funcionalidad especial al añadirle TableRow, lo que se asemejaría a un LinearLayout horizontal dentro de un LinearLayout vertical.

Se podría simular un TableLayout mediante varios LinearLayout, pero la diferencia entre uno y otro se encuentra en el tratamiento global que se le da a las vistas que se incluyen en todos los TableRow.

Este tipo de layout no solo tendría las propiedades del LinearLayout (aunque, en este caso, android:orientation carecería de sentido), sino que también tendría algunas propiedades exclusivas, como:

- *android:shrinkColumns*: permite indicar las columnas que se pueden contraer para dejar espacio al resto de columnas, adaptándose al tamaño del contenedor. Si se desea permitir que se contraigan todas las columnas de la tabla, se utilizará el valor “*”. Para igualar el aspecto de la tabla al contraer columnas, pueden utilizarse los atributos android:lines, android:scrollHorizontally o android:ellipsize.
- *android:stretchColumns*: esta propiedad nos indica las columnas que se pueden expandir para absorber el espacio libre dejado por las demás columnas incluidas en el contenedor. Los parámetros que podremos utilizar serán los mismos que los utilizados en el caso anterior.

```
<TableLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:background="#cdced4"
    android:useDefaultMargins="true">

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />
    </TableRow>

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />
    </TableRow>

```

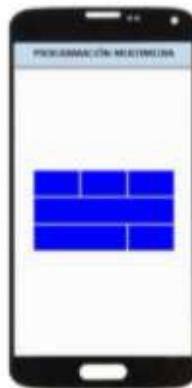


Figura 4.6

TableRow.

```

        android:layout_margin="2dp"
        android:layout_span="3"
        android:background="#0000ff" />

    <TableRow>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:layout_span="2"
            android:background="#0000ff" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="2dp"
            android:background="#0000ff" />
    </TableRow>
</TableLayout>

```

Además de lo anterior, hay otros dos interesantes parámetros aplicables a TableRow:

- *android:layout_span*: posibilita que una celda determinada pueda ocupar el espacio de varias columnas de la tabla.
- *android:layout_column*: mueve la vista a una columna diferente a la que le correspondería, según el orden en el que se ha incluido en el TableRow (las columnas se numeran a partir del 0).

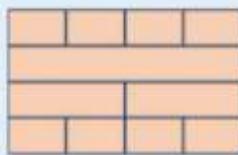
RECUERDA

- ✓ Con *android:layout_margin* se puede poner el mismo margen a los cuatro lados de una celda. Si solo se quiere hacer en uno de ellos, se puede utilizar *layout_marginBottom*, *layout_marginTop*, *layout_marginLeft*, *layout_marginRight*.
- ✓ Es posible modificar el aspecto de las celdas definiendo el Background en un fichero XML situado en los recursos (*res/xml*). En él se podrán incluir, además de colores, tipos de líneas o las formas (*shape*) con las que se presentará cada celda.

Actividad propuesta 4.4



Realiza una aplicación con una actividad cuya vista contenga un TableLayout con la estructura mostrada a la derecha. Puedes usar los botones como objetos incluidos en el interior de la tabla (siguiendo el ejemplo anterior).



4.9. GridLayout

Incluido a partir de la API 14 (Android 4.0), similar al TableLayout, utiliza un interfaz de forma tabular, distribuido en filas y columnas. A diferencia que en el TableLayout, se indica el número de filas y columnas como propiedades del layout, mediante android:rowCount y android:columnCount.

```
<GridLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:background="#cdced4"
    android:columnCount="3"
    android:orientation="horizontal"
    android:rowCount="4"
    android:useDefaultMargins="true">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#0000ff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_rowSpan="2"
        android:layout_gravity="fill"
        android:background="#0000ff" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_columnSpan="3"
        android:layout_gravity="fill"
        android:background="#0000ff" />
</GridLayout>
```

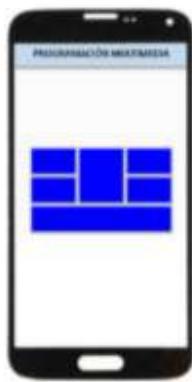
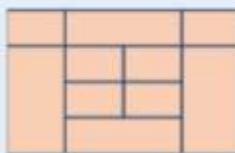


Figura 4.7
GridLayout.

- *android:layout_orientation*: marca la forma de colocar los elementos hijos, por filas o columnas.
- *android:layout_columnSpan*: posibilita que una celda pueda ocupar el espacio de varias columnas.
- *android:layout_rowSpan*: hace que una celda pueda ocupar el espacio de varias filas de la tabla.

Actividad propuesta 4.5

Realiza una aplicación con una actividad cuya vista contenga un GridLayout con la estructura mostrada a la derecha. Puedes usar los botones como objetos incluidos en el interior de la tabla (siguiendo el ejemplo anterior).



WWW

Recurso web

En este código QR encontrarás un tutorial de Shane Conder & Lauren Darcey para realizar el diseño de un teclado numérico mediante GridLayout.

**4.10. ConstraintLayout**

Con la llegada de Android Studio 2.2, Google presentó este nuevo layout con el que cambia radicalmente la forma de diseñar interfaces gráficas. ConstraintLayout permite simplificar el anidamiento de interfaces con un diseño visual para el que utiliza herramientas drag and drop.

Siguiendo el estilo del RelativeLayout, los objetos se posicionan en relación a objetos hermanos y padres, pero es más flexible y más sencillo de usar mediante la vista diseño de Android Studio. Para poder utilizar este modelo de layout se deberá incorporar su propia librería, que deberá añadirse al proyecto como una dependencia, sin embargo, Android Studio usa ConstraintLayout por defecto, por lo que esta librería ya viene incorporada a los nuevos proyectos.

WWW

Recurso web

En este código QR puedes ver el vídeo oficial de presentación de ConstraintLayout:



Se diseñan los interfaces trabajando con la vista Blueprint, que permite ver las conexiones (Constraints) entre los objetos. Al igual que en el RelativeLayout, la posición de las vistas será relativa a otro objeto de la vista o al propio layout, así siempre se ajustarán correctamente al tamaño de la pantalla, independientemente del dispositivo.

Además del ajuste de tamaño que se ha visto hasta ahora (wrap_content y match_parent), el ConstraintLayout permite ajustar las dimensiones mediante un aspect ratio.

Los cuadrados de las esquinas posibilitan cambiar el tamaño de la vista. Los círculos (manejador de restricción) permiten establecer las conexiones.

Una vista tiene que tener al menos una restricción vertical y una horizontal. Si se omite la restricción horizontal, la vista se mostrará alineada a la izquierda. Si se omite la restricción vertical, la vista se mostrará en la parte superior, independientemente de la posición de la vista en el Blueprint. Este es un ejemplo de panel de diseño en Android Studio (Design).

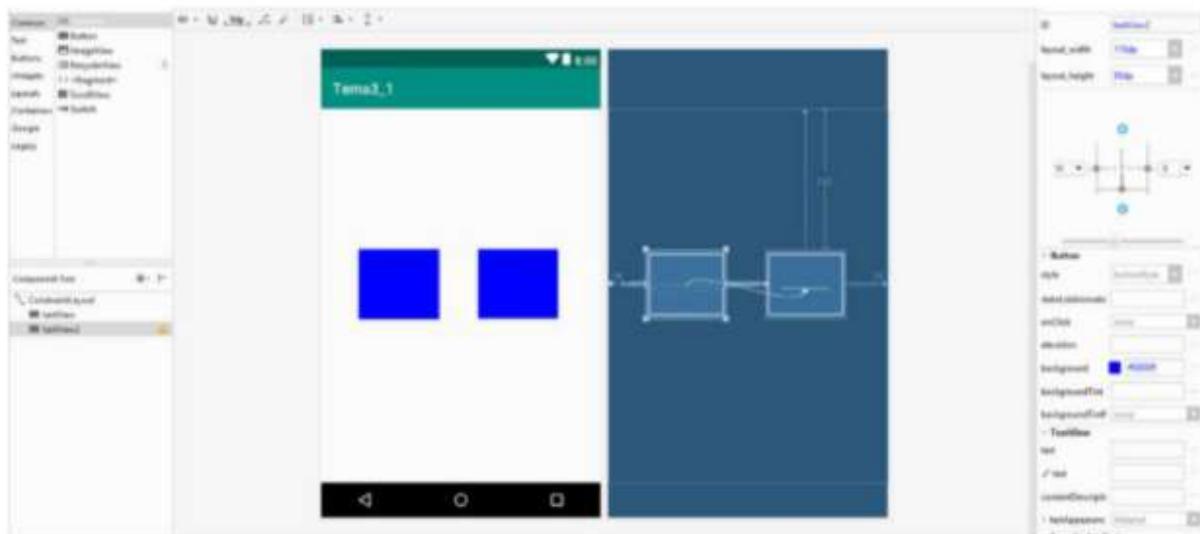


Figura 4.8
Vista diseño de ConstraintLayout en Android Studio.

La pantalla de diseño de la figura 4.8 correspondería con el siguiente código y captura de pantalla.

```
<android.support.constraint.ConstraintLayout
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/bot1"
        android:layout_width="110dp"
        android:layout_height="95dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="192dp" />

```

```

        android:layout_marginRight="16dp"
        android:background="#0000f"
        app:layout_constraintHorizontal_bias="0.825"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/boton2"
        android:layout_width="110dp"
        android:layout_height="95dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="8dp"
        android:background="#0000f"
        app:layout_constraintBaseline_toBaselineOf="@+id/boton1"
        app:layout_constraintHorizontal_bias="0.435"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@+id/boton1" />

</android.support.constraint.ConstraintLayout>

```

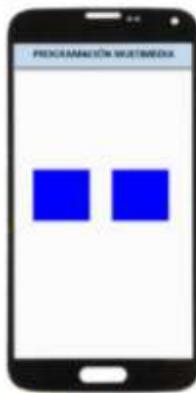


Figura 4.9
Diseño
ConstraintLayout.



PARA SABER MÁS

En el siguiente código QR encontrarás información sobre el uso del editor de diseño para layout que incorpora Android Studio.



Resumen

- Un layout es un elemento destinado a controlar la distribución, posición y dimensiones de los objetos que se ubican en su interior. Este pertenece a la clase ViewGroup que, a su vez, pertenece a la clase View.
- La clase View como clase principal tiene una serie de atributos que serán heredados por todos los elementos que de ella provienen: atributos de posicionamiento, márgenes y espaciado, entre otros.
- En función de la forma y posicionamiento en pantalla, existe gran variedad de layouts, que serán utilizados según la necesidad de cada aplicación.
- LinearLayout apila sus objetos interiores de forma horizontal o vertical, según se establezca el atributo de orientación. FrameLayouts coloca todos sus controles hijos alineados con su esquina superior izquierda, pudiendo modificar la posición con la propiedad gravity.
- AbsoluteLayout indica las coordenadas absolutas de pantalla donde queremos que se visualice cada elemento. RelativeLayout coloca los elementos que contiene, especificando la posición de cada elemento de forma relativa al elemento padre o a cualquier otro elemento incluido en el layout.

- TableLayout distribuye sus elementos hijos de forma tabular, definiendo filas, columnas y posición de cada componente dentro de la tabla. GridLayout es similar al anterior, indicándose previamente el número de filas y columnas del layout.
- Un nuevo tipo de contenedor aparece con ConstraintLayout, que permite simplificar el anidamiento mediante un interface visual que utiliza herramientas tipo drag and drop.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes términos indica al interfaz que el contenedor debe ajustarse a su tamaño?:
 - a) wrap_content.
 - b) match_parent.
 - c) fill_parent.
 - d) Ninguno de los anteriores.
2. ¿Qué atributo distribuye los objetos en el contenedor según el espacio disponible?:
 - a) layout_weight.
 - b) layout_margin.
 - c) layout_width.
 - d) layout_gravity.
3. ¿Qué atributo permite colorear el fondo de un objeto ViewGroup?:
 - a) android:background.
 - b) android:layout_weight.
 - c) android:visibility.
 - d) android:layout_below.
4. Con respecto a LinearLayout, ¿cuál de las siguientes afirmaciones es falsa?:
 - a) Apila a los objetos contenidos de forma horizontal o vertical.
 - b) La forma de apilamiento viene marcada por el atributo orientación.
 - c) No puede contener objetos de tipo ViewGroup.
 - d) Puede hacer el contenido desplazable mediante ScrollView.
5. ¿Cuál de estas afirmaciones es cierta?:
 - a) RelativeLayout posiciona a los elementos según coordenadas x-y.
 - b) AbsoluteLayout coloca a todos sus objetos en la esquina superior izquierda.
 - c) FrameLayout utiliza el atributo gravity para posicionar los objetos.
 - d) Es recomendable el uso de AbsoluteLayout para Tablets.
6. En los RelativeLayouts, ¿cuál de estas opciones no se hace en relación al control?:
 - a) android:layout_alignTop.
 - b) android:layout_centerVertical.
 - c) android:layout_toLeftOf.
 - d) android:layout_below.

7. En el TableLayout, ¿qué atributo indica las columnas que se pueden contraer?:
 a) android:ellipsize.
 b) android:shrinkColumns.
 c) android:stretchColumns.
 d) android:layout_span.
8. ¿Qué atributo permite que una celda pueda ocupar el espacio de varias columnas?:
 a) android:ellipsize.
 b) android:shrinkColumns.
 c) android:stretchColumns.
 d) android:layout_span.
9. ¿Cuál de los siguientes atributos no le aplicarías a un GridLayout?:
 a) android:rowCount.
 b) android:layout_orientation.
 c) android:layout_rowSpan.
 d) Se podrían aplicar los tres.
10. Con respecto a ConstraintLayout, ¿cuál de estas afirmaciones es falsa?:
 a) Aparece con Android Studio 1.6.
 b) Utiliza herramientas drag and drop.
 c) Trabaja con vista Blueprint.
 d) Permite ajustar las dimensiones mediante aspect ratio.

SOLUCIONES:

1. **a** **b** **c** **d**
2. **a** **b** **c** **d**
3. **a** **b** **c** **d**
4. **a** **b** **c** **d**

5. **a** **b** **c** **d**
6. **a** **b** **c** **d**
7. **a** **b** **c** **d**
8. **a** **b** **c** **d**

9. **a** **b** **c** **d**
10. **a** **b** **c** **d**

5.1. Introducción

Después de haber trabajado con los distintos tipos de layouts, en este capítulo se van a estudiar los diferentes controles que Android pone a disposición del programador para crear elegantes y funcionales interfaces. Se agruparán por la tipología del control, comenzando por los más básicos, para ver después algunos de los más complejos. Son numerosos, aun sin incluir librerías de terceras partes, por lo que difícilmente podrían ser incluidas todas sus propiedades en este libro.

5.2. TextView

Este control, de fácil construcción, quizás sea uno de los más usados, tiene la única función de mostrar un texto en pantalla. Sus parámetros más importantes, además de los comunes a otros objetos View, son:

- *android:text*. Define cuál será el texto que se mostrará en pantalla.
- *android:background*. Establece el color de fondo.
- *android:textColor*. Asigna el color del texto.
- *android:textSize*. Determina el tamaño del texto (es aconsejable utilizar unidades de medida sp).
- *android:textStyle*. Elige el estilo del texto: normal, negrita (bold) o cursiva (italic).
- *android:typeface*. Fija el tipo de letra por el que podemos optar: sans, monospace o serif.

Además de estos, disponemos de muchos otros, que seguramente se utilicen en menos ocasiones, como *android:textAppearance*, *android:shadowColor*, *android:shadowRadius*...

Para Incluirlo en el fichero de la vista (XML) de la aplicación tan solo deberá ser añadido dentro de un objeto ViewGroup (layout) de la siguiente manera:

```
<TextView  
    android:id="@+id/texto"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Texto para mostrar"  
/>
```

Para manipular este objeto de texto desde Java, primero deberá asociarse a la vista, para lo que es necesario importar la librería del widget (esto suele hacerlo de manera automática Android Studio) que da acceso a los objetos incluidos en el interface. En este caso, se necesitará *import android.widget.TextView;*

TEN EN CUENTA

- ✓ Si Android Studio deja librerías sin importar, puede hacerse de forma manual, posicionando el cursor sobre los objetos no reconocidos y pulsando Alt+Intro.

A continuación se deberá instanciar el objeto y acceder al recurso, que como ya se ha visto anteriormente, se hará de la siguiente manera:

```
TextView miTexto= (TextView) findViewById(R.id.texto);
```

Donde `miTexto` es el nombre que adquirirá esta instancia en Java y `R.id.texto` es el nombre con el que lo identificamos en el fichero XML de la vista asociada a este objeto (`android:id="@+id/texto"`).

Una vez hecho esto, se pueden obtener (get) o modificar (set) los atributos de este objeto, y así, si se quisiera cambiar el contenido de la etiqueta de texto, tan solo se tendría que escribir:

```
miTexto.setText("Nuevo texto para mostrar");
```

Un ejemplo simple podría quedar de la siguiente manera:

```
public class ActividadTexto extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.actividad_texto);
        TextView miTexto= (TextView) findViewById(R.id.texto);
        miTexto.setText("Nuevo texto para mostrar");
    }
}
```

En este ejemplo, el nombre que se le ha puesto al fichero XML, que puede ser encontrado dentro del directorio `res/layout`, ha sido `actividad_texto`.

5.2.1. Cambiar las propiedades del texto

Puede ser manipulado un gran número de propiedades de este objeto desde Java. De todas ellas, nos detendremos en la modificación del color, por las diferentes formas de uso y porque, al igual que haremos con el texto, pueden ser aplicadas a prácticamente todos los objetos de la clase `View`, lo que permite más versatilidad.

Para modificar el color de una etiqueta de texto se utilizará el comando `setTextColor()`, incluyendo entre paréntesis el color que se va a aplicar, lo que puede hacerse de variadas formas, según se muestra a continuación (en todas ellas se aplica el color rojo a la etiqueta de texto):

- `miTexto.setTextColor(0xffff0000);` directamente en hexadecimal incluyendo la transparencia (alfa).
- `miTexto.setTextColor(Color.RED);` como constante con la librería `android.graphics.Color;`
- `miTexto.setTextColor(Color.rgb(255, 00, 0));` tres dígitos en decimal (rango 0 a 255).

- `miTexto.setTextColor(Color.argb(255, 255, 0, 0));` tres dígitos en decimal con transparencia.
- `miTexto.setTextColor(Color.parseColor("#ff0000"));` nos permite obtener el color de una cadena.

Por último, con el tiempo, es aconsejable acostumbrarse, por su utilidad, a incluir los parámetros de los atributos en los recursos, lo que puede hacerse también con el color. En este caso, se procedería de la siguiente forma:

```
miTexto.setTextColor(getResources().getColor(R.color.miColor));
```

Previamente, se debe modificar el fichero `res/values/color.xml` y añadirle una nueva línea, definiendo este color personalizado.

```
<color name="miColor">#f0000</color>
```

El utilizar una u otra forma de aplicar el color, de las vistas anteriormente, dependerá de la propia práctica y, aunque parezcan suficientes, si se busca en la página de Android Developer, se podrán encontrar algunas más.

5.2.2. Añadir nuevo texto

Como se ha visto anteriormente, se puede cambiar el contenido de la etiqueta de texto tan solo con `setText`, pero si lo que se desea es añadir más texto al ya existente, se hará mediante el comando `append()`, incluyendo como parámetro el texto que se va a añadir. Si previamente se desea hacer un salto de línea, se debe incorporar el modificado `\n`. Un ejemplo podría ser:

```
miTexto.append("\n Nueva línea de texto");
```

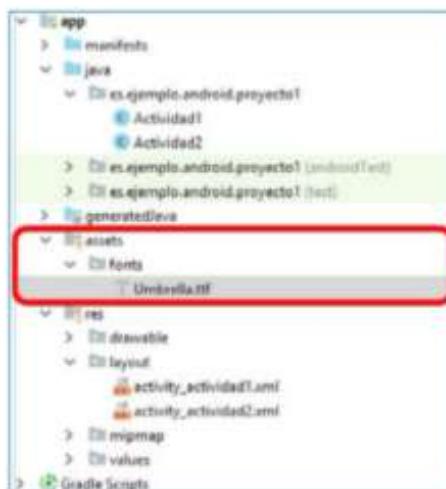
5.2.3. Uso de fuentes variadas

Para ello, se debe proceder desde Java, utilizando la librería `android.graphics.Typeface`. Las fuentes que se desean utilizar (que deberán ser true type) han de ser incluidas previamente en el directorio `assets/fonts`.

Posteriormente, se instanciará la fuente (mediante la librería antes citada) y se aplicará al texto. El resultado podría quedar así.

```
Typeface miFuente =
Typeface.createFromAsset(getAssets(),"fonts/Umbrella.ttf");
miTexto.setTypeface(miFuente);
```

Figura 5.1
Directorio de fuentes en recursos.



Actividad propuesta 5.1

Realiza las siguientes actividades de texto. Para el posicionamiento en pantalla, recuerda que debes utilizar los mismos atributos estudiados en los layouts.



Si se desarrolla para API 25 o superior se puede crear directamente la carpeta res/font (en recursos) y acceder a la fuente desde XML mediante android:fontFamily="@fonts/umbrella" o desde Java con Typeface miFuente = getResources().getFont(R.font.umbrella). En este caso, se deberán evitar las mayúsculas.

RECUERDA

- ✓ Existen varias alternativas para instanciar los objetos de texto, una sería la vista anteriormente, TextView [miTexto= [TextView] findViewById(R.id.texto);]. También podría declararse primero el objeto [TextView mitexto;] y, posteriormente, las instancias [mitexto= [TextView] findViewById(R.id.mitexto);].
- ✓ Sea cual sea la forma utilizada, se ha de tener en cuenta la visibilidad que tenga cada uno de los objetos declarado (public, private, protected).

5.2.4. Animación del texto

Android permite aplicar efectos de animación a los objetos incluidos en el interface, para lo cual pone a nuestra disposición dos librerías que lo facilitan y que, como en otras ocasiones, deberán ser importadas:

- import view.animation.Animation;
- import view.animation.AnimationUtils;

El efecto de texto que se desea aplicar se puede definir previamente en un fichero XML situado en el directorio res/anim. Este directorio no suele aparecer por defecto y posiblemente deba ser creado haciendo click con el botón derecho sobre el directorio res.

Android permite variedad de animaciones (que después se verán), pero todas ellas tienen la misma forma de estructurarse desde Java. El proceso de construcción es el siguiente:

Primero se ha de instanciar y cargar la animación.

```
Animation miAnimacion = AnimationUtils.loadAnimation(this, R.anim.animacion);
```

Donde R.anim.animacion indica la ruta y el fichero que contiene la secuencia de animación. Luego se ha de definir el modo de repetición (lo normal es que se reinicie al llegar al final) y el número de repeticiones.

```
miAnimacion.setRepeatMode(Animation.RESTART);
miAnimacion.setRepeatCount(20);
```

Por último, se aplica la animación al texto y se inicia la misma.

```
miTexto.startAnimation(miAnimacion);
```

En cuanto al tipo de animaciones que pueden ser utilizadas, las más frecuentes son:

- Translación (translate).
- Rotación (rotate).
- Escalado (scale).
- Aparición (alpha).

Según la utilizada, se tendrán que definir unos u otros parámetros (no necesariamente hay que utilizarlos todos). A modo de ejemplo: si se quisiera realizar un desplazamiento del texto por la pantalla (translate), deberán definirse el punto de origen, punto final (tanto en coordenadas X como Y), duración (en milisegundos) y la forma que se desea que tenga el movimiento (interpolación). Este último apartado es el más complejo, pero también el más efectista, ya que permite numerosas alternativas:

- linear_interpolator
- accelerate_interpolator
- decelerate_interpolator
- accelerate_decelerate_interpolator
- anticipate_interpolator
- bounce_interpolator
- overshoot_interpolator

Un ejemplo de fichero R.anim.animacion para una translación podría ser:

```
<?xml version="1.0" encoding="utf-8"?>
<translate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:fromXDelta="-100%p"
    android:toXDelta="100%p"
    android:duration="4000"
/>
```

Si se desease realizar una rotación, podría utilizarse el siguiente ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="2000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/linear_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toDegrees="360" />
```

Para crear un efecto de aparición-desaparición, podría usarse lo siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="3000"
    android:fromAlpha="0.0"
    android:interpolator="@android:anim/linear_interpolator"
    android:toAlpha="1.0" />
```

Android también permite crear secuencias de animación, que se pueden definir en el mismo fichero XML en el que se ha estado trabajando hasta ahora. El aspecto que debería tener es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    <rotate
        />
    <alpha
        />
</set>
```

Entre las correspondientes etiquetas se incluyen los parámetros deseados, así como tantas animaciones como se desee.

Por último, en el apartado animaciones también es posible definir animaciones directamente desde Java (aunque es más complejo). Para ello, se deberá instanciar la clase AnimationSet, y cada modalidad de animación que se va a emplear (AlphaAnimation, ScaleAnimation, TranslateAnimation, RotateAnimation). Posteriormente, se configura esta animación y, después, se la aplica al texto. Un ejemplo podría ser el siguiente:

```
AnimationSet animacion = new AnimationSet(true);
AlphaAnimation aparicion = new AlphaAnimation(0,1);
aparicion.setDuration(3000);
animacion.addAnimation(aparicion);
animacion.setRepeatMode(Animation.RESTART);
animacion.setRepeatCount(20);
miTexto.startAnimation(animacion);
```

En la página oficial de Android pueden consultarse los parámetros que se deben utilizar para otro tipo de animaciones.



Actividad propuesta 5.2

En la plataforma de Editorial Síntesis encontrarás diferentes ficheros de animaciones, incorpóralos a tu proyecto y observa el efecto de cada uno. También puedes modificar los atributos hasta que te acabes familiarizando con los efectos de animación.

5.3. Button

La clase `Button` va a permitir crear botones en la interfaz gráfica. Esta clase hereda de `TextView`, por lo que tiene toda su funcionalidad. Al igual que el caso anterior, puede ser creado directamente el objeto desde la vista en el fichero XML, procediendo de la siguiente forma.

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

Para manejar los eventos de este objeto (habitualmente el pulsarlo), puede hacerse de dos maneras, la primera desde XML, definiendo un evento `onClick`, al que luego accederá un método desde el fichero Java.

```
    android:onClick="pulsado"
```

El método que responderá en java debe venir definido de la siguiente forma:

```
public void pulsado (View view) {
    // Código que se ejecutará
}
```

La segunda manera, y más profesional, es mediante un manejador de eventos desde código Java. Para ello, se debe poner un identificador al botón en la vista (en este caso, `miBoton`), y luego instanciarlo y asociarle un manejador de eventos (escuchador). Sería de la siguiente forma:

```
Button miBotonJava = (Button) findViewById(R.id.miBoton);
miBotonJava.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Código que se ejecutará
    }
});
```

Algo más avanzado, sería cuando se hace que la propia Activity sea el manejador de eventos, para lo cual se ha de implementar el escuchador en la clase principal de la siguiente forma:

```
public class ClasePrincipal extends AppCompatActivity implements View.OnClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.vista_principal);
        Button miBotonJava = (Button) findViewById(R.id.miBoton);
        miBotonJava.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        // Código que se ejecutará
    }
}
```

En este caso, puede asociarse el manejador de eventos a más de un botón.

```
public class ClasePrincipal extends AppCompatActivity implements View.OnClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.vista_principal);
        Button miBoton1Java = (Button) findViewById(R.id.miBoton1);
        Button miBoton2Java = (Button) findViewById(R.id.miBoton2);
        miBoton1Java.setOnClickListener(this);
        miBoton2Java.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.miBoton1:
                // Código que se ejecutará al pulsar el botón 1
                break;
            case R.id.miBoton2:
                // Código que se ejecutará al pulsar el botón 2
                break;
            default:
                break;
        }
    }
}
```

No se ha de olvidar que deben estar previamente diseñados e identificados en el XML e importar las tres librerías necesarias para esta funcionalidad.

- import android.view.View;
- import android.view.View.OnClickListener;
- import android.widget.Button;

Investiga

Todas las vistas permiten registrar eventos de distinto tipo, entre los que tenemos los siguientes. Investiga y practica con cada uno de ellos.

`setOnClickListener`
`setOnDragListener`
`setOnFocusChangeListener`

`setOnKeyListener`
`setOnLongClickListener`
`setOnTouchListener`

**Actividad propuesta 5.3**

Realiza las siguientes actividades con botones. El objetivo es que se modifique el contenido de una etiqueta de texto tras pulsar un botón, en el primer caso, mediante un método que responde a un evento `onClick`, y el segundo caso, con un escuchador en Java.



5.4. ToggleButton

Es una variante del anterior, en el que puede personalizarse el estilo, tanto para cuando se encuentre pulsado como para cuando no lo esté.

En este ejemplo, variará el texto según la opción en la que se encuentre (si se ejecuta, se observará que, además del texto, también cambia el aspecto del botón).

```
<ToggleButton
    android:id="@+id/miToggle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textOff="NO PULSADO"
    android:textOn="PULSADO" />
```

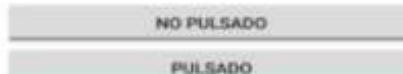


Figura 5.2
Ejemplo de `ToggleButton`.

RECUERDA

- ✓ El escuchador de este objeto podrá ser del tipo OnCheckedChangeListener() y escuchará el evento cambio de estado (onCheckedChanged).
- ✓ También puede personalizarse este botón definiendo el estilo dentro del fichero res/values/styles, incluso creando un recurso XML con el diseño personalizado.

5.5. ImageButton

Es otro objeto de interface que hereda de ImageView (se estudiará más adelante), que permite crear botones ilustrados con una imagen (que debe previamente ser incluida en la carpeta res/drawable), sin embargo, a estos no es posible ponerle texto. Tanto para este como para el anterior, puede utilizarse cualquiera de los manejadores de eventos estudiados con Button.

```
<ImageButton
    android:id="@+id/miBoton"
    android:src="@drawable/imagen"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="pulsado" />
```

5.6. EditText

Es el objeto más simple que nos ofrece Android para leer entradas de texto. Este objeto hereda todas las propiedades de TextView.

```
<EditText
    android:id="@+id/miTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Para manipular este objeto desde Java, se necesita el widget incluido en la librería android.widget.EditText, y se deberá proceder de la siguiente manera:

```
EditText textoLeido = (EditText) findViewById(R.id.miTexto);
String cadenaTexto = textoLeido.getText().toString();
```

Por defecto, EditText lee texto plano y asocia el teclado textual del dispositivo móvil, pero dispone de numerosas opciones, gracias al modificador keyboardType, según la conveniencia de la aplicación desarrollada.

- *textUri*. Texto que se usará como URI.
- *textEmailAddress*. Texto que se usará como dirección de correo.
- *textPersonName*. Nombre de una persona.

- *textPassword*. Contraseña.
- *textVisiblePassword*. Contraseña que se mostrará.
- *number*. Entrada numérica.
- *phone*. Entrada de un número de teléfono.
- *date*. Texto tratado como fecha.
- *time*. Texto tratado como hora.
- *textMultiLine*. Permite multilinea.
- *textCapSentences*. Pone en mayúsculas la primera letra de cada frase.
- *textCapWords*. Pone en mayúsculas la primera letra de cada palabra.
- *textNoSuggestions*. Desactiva la escritura predictiva.

TOMA NOTA



En caso de querer utilizar más de una opción, se utilizará el separador |.

Otras interesantes opciones, que posiblemente se usen en más de una ocasión, son:

- *android:hint*. Pone un texto predefinido (que aparecerá en gris).
- *android:lines*. Limita el número de líneas.
- *android:digits*. Limita el número de dígitos (numérico).
- *android:maxLength*. Limita el número de caracteres.

Investiga



Algo más compleja sería la opción *android:imeOptions*, que define el botón de opción del teclado entre las siguientes posibilidades. Investiga y practica con cada uno de ellos.

<i>actionNone</i> : el botón muestra return	<i>actionDone</i> : ok
<i>actionGo</i> : ir	<i>actionNext</i> : siguiente
<i>actionSearch</i> : búsqueda	<i>actionPrevious</i> : anterior
<i>actionSend</i> : enviar	<i>actionUnspecified</i> : no especificada

Un ejemplo de todo lo anterior podría ser el siguiente:

```
<EditText
    android:id="@+id/correo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Avenida de Madrid"
    android:imeOptions="actionSearch"
    android:inputType="textPostalAddress|textCapWords|textNoSuggestions" />
```

Para manejar los eventos del EditText (cambios de texto mientras escribimos), se dispone de varios métodos que responden en diferentes momentos.

```
textoLeido.addTextChangedListener(new TextWatcher() {
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        // Nos responde cuando el texto está cambiando
    }
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        // Nos responde antes de que el texto cambie
    }
    @Override
    public void afterTextChanged(Editable s) {
        // Nos responde después de que el texto cambie
    }
});
```

Por último, para manejar los eventos del botón de acción declarado en el teclado virtual a través del atributo android:imeOptions se deberá usar el escuchador OnEditorActionListener junto a su controlador onEditorAction(). Un ejemplo para imeOptions de búsqueda sería así:

```
textoLeido.setOnEditorActionListener(new TextView.OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        if (actionId == EditorInfo.IME_ACTION_SEARCH) {
            // Código que se ejecutará al pulsar el botón de acción
            return true;
        }
        return false;
    }
});
```

Actividad propuesta 5.4



Implementa un EditText en tu proyecto y modifica los parámetros para obtener los siguientes tipos de teclados:



5.7. AutoCompleteTextView

Este objeto es una variante del anterior, al que se añaden sugerencias de autocompletado. Para ello, tiene una importante propiedad android:completionThreshold, que define el número de caracteres mínimo que se han de escribir para que la aplicación ofrezca una sugerencia (en este ejemplo, cuatro caracteres).

```
<AutoCompleteTextView
    android:id="@+id/miTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:completionHint="Elige una opción"
    android:completionThreshold="4" />
```

Los elementos que se sugerirán deben ser definidos en el fichero Java en un array de string, al que se le debe crear un adaptador del tipo ArrayAdapter (más adelante se estudiarán los adaptadores en profundidad), cuya misión es adaptar los datos del array al AutoCompleteTextView. Este adaptador se pasará como argumento en el constructor, junto al contexto de la actividad y la forma de mostrarse (en este caso, en forma de lista desplegable).

Por último, se pasa el adaptador a la instancia del texto definido en el XML. Como en casos anteriores, deberán importarse las librerías necesarias, que aquí serán android.widget.ArrayAdapter y android.widget.AutoCompleteTextView). El código final Java quedaría así:

```
String[] opciones = {"Opción 1", "Opción 2", "Opción 3", "Opción 4", "Opción 5"};
AutoCompleteTextView textoLeido = (AutoCompleteTextView) findViewById(R.id.miTexto);
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, opciones);
textoLeido.setAdapter(adaptador);
```

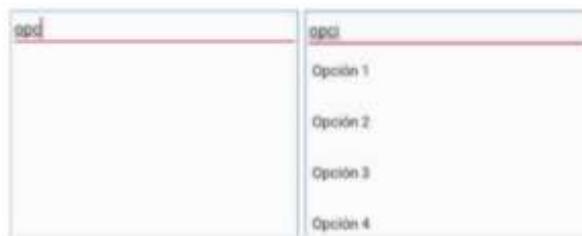


Figura 5.3
Autocompletado con tres caracteres.

5.8. MultiAutoCompleteTextView

La diferencia entre este objeto y el anterior es que este permite ofrecer sugerencias para cada bloque de texto introducido, marcando mediante un delimitador (token) el elemento a partir del cual volverá a hacernos la sugerencia. En el ejemplo siguiente se utiliza la coma como elemento delimitador.

```
MultiAutoCompleteTextView textoLeido = (MultiAutoCompleteTextView) this.findViewById(R.id.miTexto);
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, opciones);
textoLeido.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
textoLeido.setAdapter(adaptador);
```

5.9. Spinner

El widget Spinner de Android muestra una lista desplegable para seleccionar un único elemento, y es equivalente a ComboBox de otros lenguajes de programación.

```
<Spinner
    android:id="@+id/miSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

En Java este objeto depende de la librería android.widget.Spinner, y para su uso deberá instanciarse, asignarle un array de valores y un adaptador (del tipo simple_spinner_item).

```
Spinner spinner = (Spinner) findViewById(R.id.miSpinner);
String[] valores = {"Valor1", "Valor2", "Valor3", "Valor4", "Valor5"};
spinner.setAdapter(new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item, valores));
```

También puede añadirse la lista de valores directamente en el XML mediante el comando android:entries="@array/valores", debiéndose crear previamente el fichero res/values/valores.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="valores">
        <item>Valor1</item>
        <item>Valor2</item>
        <item>Valor3</item>
        <item>Valor4</item>
        <item>Valor5</item>
    </string-array> </resources>
```



Figura 5.4
Panel de Spinner desplegado.

El adaptador en Java se ha de aplicar de la siguiente manera:

```
Spinner spinner = (Spinner) findViewById(R.id.miSpinner);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.valores, android.R.layout.simple_spinner_item);
spinner.setAdapter(adapter);
```

Para leer el elemento seleccionado, se escribirá el siguiente código:

```
String value = spinner.getSelectedItem().toString();
```

Para realizar un escuchador que responda a los eventos de cambio, se utiliza el setOnItemSelectedListener, que puede responder a dos eventos, onItemSelected y onNothingSelected. Este devolverá el contenido del array, que se encuentra en la posición tocada del adaptador.

```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> adaptador, View view, int posicion, long id) {
        adaptador.getItemAtPosition(posicion);
    }
    public void onNothingSelected(AdapterView<?> adaptador) {
    }
});
```

Como ya se vio anteriormente, puede ser implementado el escuchador en la clase principal.

5.10. CheckBox

Un control CheckBox se suele utilizar para marcar o desmarcar opciones en una aplicación. En Android está representado por la clase del mismo nombre, CheckBox.

```
<CheckBox
    android:id="@+id/miCheckbox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Pulsa Opción"
    android:checked="false"/>
```

La posición de partida es definida por el parámetro android:checked. Al igual que con los botones, puede utilizarse la opción onClick para que un método responda en Java al evento pulsado.

```
    android:onClick="tocado"
```

En este caso, el código que debería incluirse en la aplicación sería el siguiente:

```
public void tocado (View view){
    boolean pulsado = ((CheckBox) view).isChecked();
    if (pulsado) {
        //Código para pulsado
    } else {
        //Código para no pulsado
    }
}
```

Más profesional es que se use un manejador de eventos en Java, basado en setOnCheckedChangeListener y en onCheckedChanged, como puede verse a continuación:

```
CheckBox checkBox = (CheckBox) findViewById(R.id.micheckbox);
checkBox.setOnCheckedChangeListener(
    new CheckBox.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean pulsado) {
            if (pulsado) {
                //Código para pulsado
            } else {
                //Código para no pulsado
            }
        }
    });
});
```

5.11. RadioButton

Es un control de selección para elegir una opción de entre un conjunto. Se trata de un control de exclusión mutua, es decir, la selección de uno implica el cambio automático del que previamente estaba seleccionado. Pueden agruparse mediante RadioGroup, de tal manera que, de los que estén incluidos en este grupo, solo uno de ellos puede y debe estar marcado.

```
<RadioGroup
    android:id="@+id/grupo"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <RadioButton
        android:id="@+id/radiol"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1" />
    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 2" />
</RadioGroup>
```

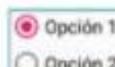


Figura 5.5
Ejemplo de RadioButton.

Puede desmarcarse (clearCheck) o marcarse (check), y localizar el elemento marcado, como ocurre en este ejemplo:

```
RadioGroup miGrupo = (RadioGroup) findViewById(R.id.grupo);
miGrupo.clearCheck();
miGrupo.check(R.id.radiol);
int idMarcado = miGrupo.getCheckedRadioButtonId();
```

Y al igual que en el CheckBox, puede asociarle mediante android:onClick un método (uno a uno) que lo controle desde Java o bien asociarle un manejador de eventos al grupo en su conjunto.

```
RadioGroup miGrupo = (RadioGroup)findViewById(R.id.grupo);
miGrupo.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        //Código para realizar
    }
});
```



Actividad propuesta 5.5

Realiza un proyecto con los RadioButton que aparecen en la imagen y que respondan al evento checked cambiando una etiqueta de texto.



5.12. Switch

Este objeto puede utilizarse en el diseño del interfaz y consta de dos estados, encendido (marcado) o apagado (desmarcado), bastante parecido al ToggleButton (ambos subclase de CompoundButton), pero lo diferencia en que este simula un botón con control deslizante.

```
<Switch
    android:id="@+id/miswitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Para manejar los eventos, debe procederse de una manera similar que en anteriores ocasiones.

```
switch pulsador = (Switch) findViewById(R.id.miswitch);
pulsador.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean pulsado) {
        if (pulsado) { //código para pulsado
        } else { //código para no pulsado
    }
});
```



Investiga

Los tres últimos objetos que hemos estudiado, CheckBox, RadioButton y Switch, pueden mejorar mucho su aspecto gráfico mediante su personalización, trabajando con los ficheros de estilo en los recursos. Investiga y practica con cada uno de ellos.



Figura 5.6
Estilos de Switch.

5.13. SeekBar

Este control pertenece al tipo de controles de selección en los que el usuario elige un valor numérico entre un rango de valores predefinidos en la aplicación. Para el diseño en XML se dispone de interesantes atributos.

- *android:max*. Define el valor máximo del control.
- *android:min*. Define el valor mínimo del control.
- *android:progress*. Fija el punto de partida del control.
- *android:thumb*. Personaliza la imagen del elemento deslizable.
- *android:rotation*. Permite girar el control y ponerlo verticalmente.
- *android:progressDrawable*. Personaliza el aspecto de la barra de desplazamiento.

```
<SeekBar
    android:id="@+id/miseekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="10"
    android:progress="0"/>
```

Figura 5.7
Ejemplo
de SeekBar.



Este objeto posee un escuchador que nos permite manejar tres eventos, uno para cuando empezamos a mover el control (*onStartTrackingTouch*), durante el movimiento (*onProgressChanged*) y al dejar de actuar sobre él (*onStopTrackingTouch*).

```
SeekBar miControl = (SeekBar) findViewById(R.id.miseekbar);
miControl.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        //Código para cambio de valor
    }
    public void onStartTrackingTouch(SeekBar seekBar) {
        //Código para inicio de cambio
    }
    public void onStopTrackingTouch(SeekBar seekBar) {
        //Código para final de cambio
    }
});
```

Este control, al igual que todos los demás, tiene multitud de posibilidades de configuración, especialmente en su aspecto estético.

Investiga



Este objeto dispone de un estilo propio que nos permite ver cada una de las fracciones de avance `style="@style/Widget.AppCompat.SeekBar.Discrete"`, que nos dejaría el siguiente aspecto:

Figura 5.8
Ejemplo de SeekBar con fracción de avance.



Además de poder crear tu estilo propio de barra en los recursos, puedes modificar el color tan solo con los siguientes atributos.

Figura 5.9
Ejemplo de SeekBar con estilo de color.



5.14. RatingBar

Es otro control de selección, pero aquí, como su nombre indica, está diseñado para dar una puntuación. Los atributos de este control son múltiples y sería conveniente consultar la página oficial de desarrolladores de Android. En el siguiente ejemplo, se fijan el número de estrella y el valor de los incrementos. En algunos dispositivos, el número de estrellas debe ajustarse a los márgenes establecidos para este control.

```
<RatingBar
    android:id="@+id/miRating"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:numStars="7"
    android:rating="1.0"
    android:stepSize="1.0" />
```



Figura 5.10
Ejemplo de RatingBar.

El manejador de eventos que debe ser utilizado será de la siguiente forma:

```
RatingBar miControl = (RatingBar) findViewById(R.id.miRating);
miControl.setOnRatingChangeListener(new RatingBar.OnRatingChangeListener() {
    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean b) {
        //Código para cambio de valor
    }
});
```

5.15. ProgressBar

Un interesante control que suele utilizarse con las tareas asíncronas y que sirve para informar al usuario de la evolución de un proceso, rellenando el tiempo de espera entre la acción del

usuario y la respuesta de la aplicación (por ejemplo, cargar una página web, almacenar un fichero...). En este control existen dos posibilidades de diseño, cuyo código se muestra a continuación.

A) Barra de progreso circular

```
<ProgressBar
    android:id="@+id/miProgreso"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyle"/>
```

B) Barra de progreso lineal

```
<ProgressBar
    android:id="@+id/miProgreso"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleHorizontal"
    android:max="100"/>
```

WWW

Recurso web

En el siguiente vídeo de Android para Principiantes encontrarás un tutorial para construir una aplicación con ProgressBar.



Resumen

- Android dispone de numerosos controles básicos para el diseño de la interface. Todos ellos, al provenir de la clase View, heredan las propiedades de posicionamiento, aspecto, márgenes..., y pueden ser construidos desde la vista en XML o de manera dinámica desde el Java.
- TextView es un control de uso frecuente con numerosos atributos, cuya función es mostrar texto en pantalla. Este, al igual que muchos de los demás, permite que le sean aplicadas animaciones.
- Button es utilizado para crear botones en la interfaz gráfica. Es importante el manejo de eventos de este control, que puede hacerse utilizando un método en Java, llamado desde el XML o mediante un escuchador en Java. Este control tiene variantes, como

el ToggleButton, con diferente aspecto según su estado; ImageButton, que permite ser ilustrado con una imagen; o Switch, subclase de CompoundButton, similar al anterior, pero que simula un botón con control deslizante.

- Como control de captación de texto se dispone de EditText, también con un numeroso grupo de atributos que le otorgan grandes posibilidades de configuración. Variantes de este son el AutoCompleteTextView, que ofrece sugerencias a la hora de introducir texto, o MultiAutoCompleteTextView, similar al anterior, aunque en este caso las sugerencias las hace por bloque de texto.
- Entre los controles de selección, se puede optar por el Spinner, en el que se realiza la selección a partir de un grupo de opciones que se muestran en una lista desplegable; CheckBox, un control para marcar o desmarcar opciones; o RadioButton, que limita la selección exclusivamente a un elemento.
- Entre los controles gráficos, se dispone de SeekBar, donde el usuario elige un valor numérico entre un rango de valores predefinidos; RatingBar, también de selección, pero diseñado para dar una puntuación; y, por último, ProgressBar, control que se utiliza fundamentalmente para informar al usuario de la evolución de un proceso.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes comandos nos permite añadir texto a una etiqueta TextView?:
 a) setText.
 b) append.
 c) setTypeface.
 d) textAppearance.
2. ¿Cuál de estas animaciones no está definida en Android?:
 a) translate.
 b) rotate.
 c) entrate.
 d) scale.
3. ¿Cómo podemos detectar un pulsado de botón en Android?:
 a) Mediante un método definido con android:onClick en el XML.
 b) Instanciando el botón en Java y con un escuchador tipo setOnClickListener.
 c) Implementando un escuchador en la clase y asociando el evento al botón.
 d) Con cualquiera de los anteriores.
4. ¿Cuál de los siguientes objetos es deslizante?:
 a) Button.
 b) ToggleButton.
 c) ImageButton.
 d) Switch.

5. ¿Qué `inputType` pondrías en un `EditText` para desactivar la escritura predictiva?:
 a) `textUri`.
 b) `textPersonName`.
 c) `textNoSuggestions`.
 d) `textCapSentences`.
6. ¿Qué atributo define el número de caracteres que se han de escribir para hacer una sugerencia de texto en `AutoCompleteTextView`?:
 a) `android:completionHint`.
 b) `android:completionThreshold`.
 c) `android:imeOptions`.
 d) `android:toAlpha`.
7. ¿Qué utilizaremos para diferenciar los bloques de texto en `MultiAutoCompleteTextView`?:
 a) Un adaptador.
 b) Un token.
 c) Un espacio en blanco.
 d) Las comillas.
8. ¿Cuál de estos adaptadores nos ofrece Android para utilizar en un `Spinner`?:
 a) `simple_spinner_item`.
 b) `simple_item_spinner`.
 c) `simple_spinner_adapter`.
 d) `simple_adapter_spinner`.
9. ¿Qué método nos responde cuando estamos moviendo un `SeekBar`?:
 a) `onStartTrackingTouch`.
 b) `onProgressChanged`.
 c) `onStopTrackingTouch`.
 d) `onSeekBarChangeListener`.
10. ¿Cuál de estos objetos utilizarías en espera de la carga de una página web?:
 a) `Switch`.
 b) `SeekBar`.
 c) `RatingBar`.
 d) `ProgressBar`.

SOLUCIONES:

1. a b c d
 2. a b c d
 3. a b c d
 4. a b c d

5. a b c d
 6. a b c d
 7. a b c d
 8. a b c d

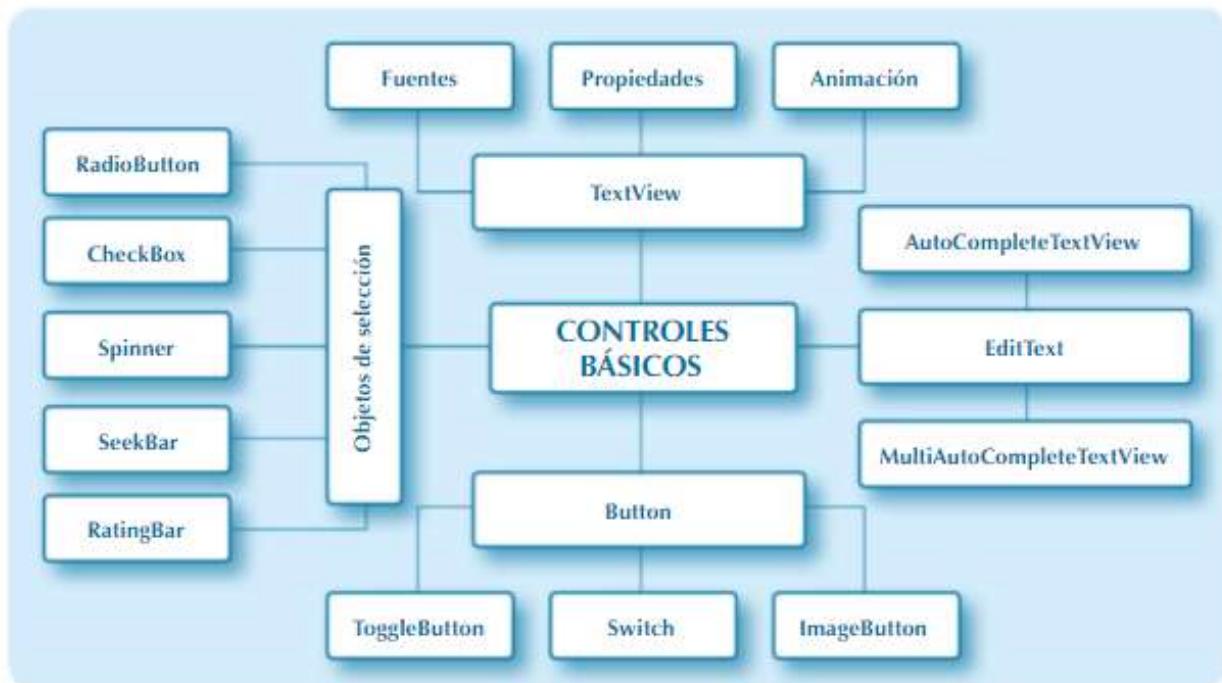
9. a b c d
 10. a b c d

Interface de usuario: controles básicos

Objetivos

- ✓ Conocer los controles básicos de Android.
- ✓ Manejar con soltura las etiquetas de texto y sus atributos.
- ✓ Identificar las posibilidades de los diferentes tipos de botones.
- ✓ Afianzarse en el uso de manejadores de eventos.
- ✓ Saber utilizar las capturas de texto y sus variantes.
- ✓ Dar adecuado uso a los objetos de selección y manejar sus eventos.
- ✓ Comprender el funcionamiento de los adaptadores en los Spinner.
- ✓ Practicar con objetos de selección gráfica (SeekBar, RatingBar...).
- ✓ Familiarizarse con los efectos producidos por los atributos de animación.

Mapa conceptual



Glosario

Adaptador. Es un objeto de una clase que actúa como puente de enlace en la presentación visual entre un conjunto de datos y la vista de los mismos.

ARGB. Acrónimo que se utiliza para establecer u obtener la propiedad del color de un objeto gráfico, definido por un componente de rojo (R), un componente de verde (G) y un componente de azul (B), y un factor de opacidad (alfa).

Assets. Carpeta de recursos de Android cuyo contenido es agregado al generar el paquete .apk, conservando su nombre y características, y permitiendo el uso de signos y mayúsculas.

findViewById. Método usado en Android cuya función es recorrer el árbol de vistas en busca del recurso que tiene la identificación aportada.

imeOptions. Posibilidad de algunos objetos del interfaz de Android que permiten adaptar el teclado de entrada según la funcionalidad y uso que se le vaya a dar al objeto.

Interpolación. Algoritmo matemático utilizado en el grafismo y la animación de Android que permite la obtención de secuencias intermedias a partir de una inicial y otra final.

Widget. Pequeñas aplicaciones en forma de subclases que facilitan las tareas de codificación de los componentes de la interfaz en Android.

Listados y menús

Objetivos

- ✓ Identificar el uso de los diferentes tipos de menús.
- ✓ Distinguir la creación de menús mediante XML y mediante Java.
- ✓ Preparar la respuesta a la opción seleccionada entre las ofrecidas en el menú.
- ✓ Adecuar el uso de cada variante de menú a las necesidades de la aplicación.
- ✓ Entender la arquitectura de los listados en Android.
- ✓ Manejar con soltura los atributos en ListView, GridView y Spinner.
- ✓ Comprender la construcción de los manejadores de eventos de estos objetos.
- ✓ Conocer los adaptadores, sus variantes, función, uso y construcción.
- ✓ Utilizar adaptadores, personalizándolos según los objetos que necesite el listado.
- ✓ Dar sentido estético y funcional al uso de listados dentro de la aplicación.

Mapa conceptual



Glosario

ArrayAdapter. Tipo de adaptador que maneja una lista o matriz de objetos como entrada, devolviendo cada elemento de la colección como una vista.

BaseAdapter. Clase base común (que puede ser personalizada) de implementación general de un adaptador para su uso en ListView, GridView, Spinner, etc.

Contexto. Definimos así la situación actual del objeto o aplicación. Nos provee de información sobre su actividad, permitiéndonos el acceso a clases y recursos específicos de la aplicación.

Dirty View. Vista secundaria en proceso de entrada en pantalla, que el adaptador debe preparar antes de mostrarse.

Inflar. En Android se denomina así a la acción mediante la cual se agrega una jerarquía de vistas a la actividad en tiempo de ejecución.

Menú contextual. Ventana con un grupo de opciones que se abre cuando hacemos click sobre un objeto, generalmente con el botón secundario del ratón o con una pulsación sostenida en un dispositivo móvil.

POJO (Plain Old Java Object). Instancia de una clase que ni extiende ni implementa nada. Podemos utilizar objetos de este tipo relacionándolos con el adaptador para enlazar el componente del interfaz con la colección de datos.

ScrapViews. Vista secundaria que ha salido de pantalla, se mantiene en memoria y puede volverse a usar, posteriormente, tan solo actualizándose (reciclando).

6.1. Introducción

Tras el trabajo con controles básicos y después de haber estudiado los layouts, en este capítulo se va a avanzar en objetos ViewGroup. Debemos recordar que estos objetos son utilizados para la creación de una jerarquía de objetos View. Pueden ser instanciados de igual manera que los vistos anteriormente, desde XML o en código Java.

Un número importante de ellos son utilizados para mostrar la información al usuario en forma de lista (con variaciones visuales); son un conjunto de elementos ordenados por los que es posible navegar utilizando una barra de Scroll. Al seleccionar o pulsar uno de los elementos de la lista, se desencadenará un evento, que es posible controlar y al que se puede responder.

Para dar un aspecto personalizado a los listados en cualquiera de sus variantes se deberá hacer uso de los adaptadores, que permitirán aumentar el número de objetos que se van a incluir en cada uno de los elementos que contiene el listado, así como realizar un diseño estéticamente a medida del que se desea que tengan dichos elementos.

Por otro lado, como cualquier entorno que disponga de interfaz gráfica, los menús son de extraordinaria utilidad en los procesos de selección entre opciones. Android tiene variantes de este tipo de objetos, algunas ya desaprobadas y otras que han evolucionado especialmente con la llegada de Material Design. Aquí se verán algunos de los más importantes.

6.2. Los listados

En este punto del capítulo, se aprenderá a crear listas en Android y a manejar sus diferentes elementos, trabajando con tres de los objetos ViewGroup más frecuentes. Además, se iniciará el estudio en el interesante mundo de los adaptadores. En la figura 6.1 se muestra cuál sería el aspecto que tendría cada uno de estos objetos si se sigue el código de cada apartado.



Figura 6.1
Tipos de listados
en Android.

6.3. ListView

Este objeto visualiza una lista deslizable verticalmente de varios elementos; cada uno de ellos puede ser seleccionado sobre el propio control. En caso de disponer de más elementos de los que es posible mostrar en pantalla, aparecerá una barra de Scroll que permitirá acceder a los elementos no visibles. El código XML del objeto sobre el que construir el listado es el siguiente.

```
<ListView android:id="@+id/miLista"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```



Investiga

Podemos utilizar también los siguientes atributos para modificar la estética del ListView. Practica con cada uno de ellos.

- `android:divider`: permite configurar el color de la línea divisoria.
- `android:dividerHeight`: configura el grosor de la línea divisoria.
- `android:footerDividersEnabled`: habilita el divisor que va delante del "footer" (último).
- `android:headerDividersEnabled`: habilita el divisor que va delante del "header" (primero).

Los pasos que se van a seguir para poder manejar la lista desde el código Java son: instanciar el objeto, declarar un array de valores (string), crear un adaptador que nos permita llenar la lista y asociar el adaptador a la lista.

```
ListView listado = (ListView) findViewById(R.id.miLista);
final String[] datos = new String[]{"Elemento1", "Elemento2", "Elemento3", "Elemento4", "Elemento5"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, datos);
listado.setAdapter(adaptador);
```



TOMA NOTA

Un adaptador suele ser un objeto de la subclase `BaseAdapter`, que posibilita el convertir los datos en diferentes elementos de la lista. Android proporciona algunos adaptadores estándar, como `ArrayAdapter` y `CursorAdapter`. El primero permite manejar datos cuyo origen es un array, mientras que en el segundo caso los datos provendrán de una base de datos.

Para obtener la información del elemento pulsado se utiliza el manejador de eventos `setOnItemClickListener`, que podrá controlar el evento `onItemClick`, al que se le pasará como parámetro el adaptador y devolverá la posición del elemento tocado.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int posicion, long id) {
        //obtener el texto del elemento pulsado
    }
});
```

Una vez se tenga esta posición, se puede obtener el contenido del elemento en sí mediante dos formas, una de ellas pidiendo al listado (`parent`) que devuelva el elemento que se encuentra en la posición tocada, como se muestra en este ejemplo:

```
String elemento = (String) parent.getItemAtPosition(posicion);
```

O bien obteniendo el adaptador del listado y pidiéndoselo a este:

```
String elemento = (String) parent.getAdapter().getItem(posicion);
```

RECUERDA

- ✓ Un método importante de los adaptadores es el `getView`, ya que lo más frecuente es que todos los elementos de un listado no cojan en pantalla (en la vista), por lo que el adaptador se ocupará de reutilizar filas y llenar sus vistas asociadas (cada fila de la lista será una vista) con los nuevos datos para mostrar.



Actividad propuesta 6.1

Realiza una aplicación cuya actividad (Activity) tenga objetos dentro de un LinearLayout vertical, el primero un ListView y el segundo un TextView. Rellena el ListView con el nombre de diez países europeos. Cuando se ejecute el TextView, debe tomar el nombre del elemento tocado del listado.

6.4. GridView

Muestra los datos a modo de rejilla bidimensional e incluye automáticamente un Scroll para cuando los datos ocupen más tamaño que las posibilidades de la pantalla. Al igual que en el ListView, los datos provienen de un ListAdapter o incluso de un Adapter personalizado. Atributos importantes para este objeto son:

- `android:numColumns`: número de columnas que deseamos establecer en el GridView.
- `android:columnWidth`: define el ancho de cada columna de la cuadrícula.
- `android:verticalSpacing`: separación vertical entre las filas del GridView.
- `android:horizontalSpacing`: separación horizontal entre las columnas del GridView.

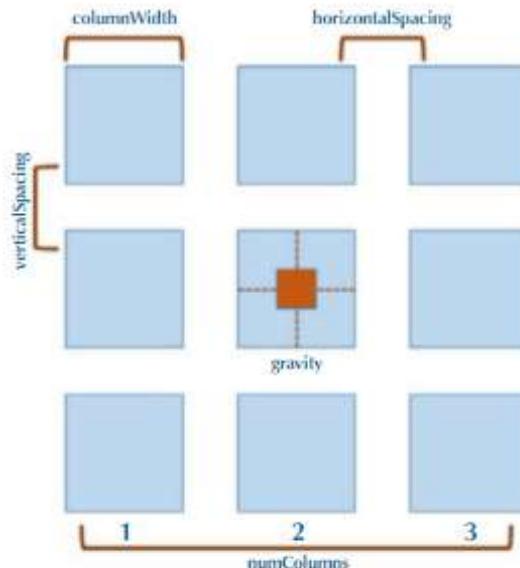


Figura 6.2
Esquema de organización de GridView.

- *android:stretchMode*: define el modo en que se extenderán las columnas.
- *android:gravity*: define el posicionamiento del contenido en cada celda.



Investiga

Las posibilidades de stretchMode son:

- *None*: ninguna extensión.
- *spacingWidth*: se extiende al espacio entre cada columna.
- *columnWidth*: se hace un reparto equitativo del espacio.
- *spacingWidthUniform*: se hace un reparto uniforme.

Otra posibilidad es usar el flag “*auto_fit*” que nos ajustará el número de columnas dependiendo de las dimensiones de la pantalla de nuestro dispositivo.

Investiga y practica con cada uno de ellos.

El código XML del objeto sobre el que construir el listado es similar al anterior, pero en este caso se añadirá el atributo del número de columnas que se desea que tenga la rejilla.

```
<GridView
    android:id="@+id/miGrid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="2"/>
```

Para manejar el GridView desde Java tras instanciar el objeto y declarar el array de valores (string), se debe asociar un adaptador a la lista.

```
GridView listado = (GridView) findViewById(R.id.miGrid);
final String[] datos = new String[]{"Elemento1", "Elemento2", "Elemento3", "Elemento4", "Elemento5"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, datos);
listado.setAdapter(adaptador);
```

Y para detectar el elemento tocado sería exactamente igual que en el ListView.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int posicion, long id) {
        //Obtener el texto del elemento pulsado
    }
});
```



Actividad propuesta 6.2

Modifica el código de la actividad realizada para ListView cambiando este objeto por un GridView de dos columnas, que tenga igual respuesta al tocar uno de los elementos.

6.5. Spinner

Este objeto es una evolución del estudiado entre los objetos básicos del interface, tan solo que su aspecto puede ser mejorado mediante adaptadores personalizados. Similar en la construcción al anterior, este objeto muestra los datos en un listado desplegable de elementos, de los cuales el usuario puede seleccionar uno, que se mostraría al frente del Spinner.

```
<Spinner
    android:id="@+id/miSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Como en otras ocasiones, para manejar el Spinner desde Java se debe instanciar el objeto, declarar el array de valores (string) y asociarle un adaptador a la lista.

```
Spinner listado = (Spinner) findViewById(R.id.miSpinner);
final String[] datos = new String[]{"Elemento1", "Elemento2", "Elemento3", "Elemento4", "Elemento5"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, datos);
listado.setAdapter(adaptador);
```

Con un escuchador del tipo setOnItemSelectedListener se detectará el elemento pulsado.

```
listado.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position, long id)
    {
        //Obtener el texto del elemento pulsado
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent)
    {
        //Obtener el texto del elemento pulsado
    }
});
```



Actividad propuesta 6.3

Realiza una aplicación similar a las dos anteriores, pero con Spinner.

6.6. Trabajando con adaptadores

Si bien la utilización del simple_list_item_1 como adaptador es bastante sencilla, si se necesitara incluir más de un elemento en el listado o se quisiera mejorar el aspecto visual del mismo (personalizando colores, fuentes o mostrando imágenes, enlaces web...), se tendría que elaborar un adaptador personalizado. En este apartado se va a aprender cómo funcionan los adaptadores y a construirlos en algunas de sus variantes.

El entender la función de los adaptadores hará más fácil su programación e implementación en las aplicaciones. Para ello, se va a ver cómo funciona el reciclaje de las vistas.

Cuando se conecta un ListView a un adaptador, este se encarga de crear las instancias de las filas necesarias hasta que ListView tenga suficientes elementos como para completar la altura que ocupa el listado en la pantalla, no siendo necesario preparar elementos adicionales en memoria.



Figura 6.3
Función del adaptador.

Si se interacciona con el listado y se desplazan los elementos hacia arriba o hacia abajo, aquellos elementos que salen de la pantalla quedarán en memoria para un uso posterior, incorporando tantas nuevas filas como las que han sido guardadas en memoria.

Las vistas que salen, denominadas *Scrap Views*, pueden volverse a utilizar posteriormente sin tener que inflarse, solo deben actualizarse (reciclarse), por tanto, el objeto de lista solo necesita mantener suficientes vistas en la memoria para completar el espacio asignado en el diseño, y algunas vistas reciclables adicionales.

Tras estos conceptos básicos sobre adaptadores, se va a crear uno que gestione un ListView (útil también para GridView y Spinner), que mostrará dos líneas de texto con formato personalizado.

El primer paso será realizar una clase Java que gestione y suministre los datos. Esta deberá tener tantos atributos como se vayan a poner en el listado. Para este ejemplo podría servir la siguiente:

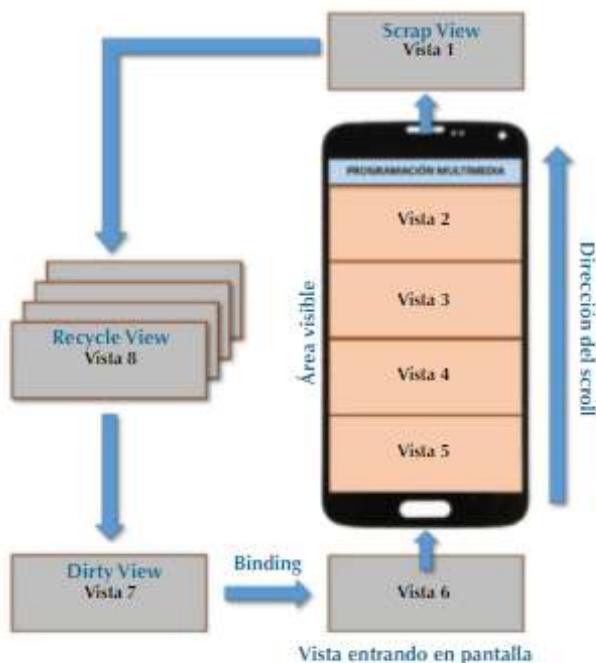


Figura 6.4
Dinámica de las vistas en pantalla.

```

public class Datos {
    private String texto1;
    private String texto2;

    public Datos (String text1, String text2){
        texto1 = text1;
        texto2 = text2;
    }
    public String getTexto1(){
        return texto1;
    }
    public String getTexto2(){
        return texto2;
    }
}

```



Figura 6.5
Ejemplo
de adaptador
con dos ejemplos.

A continuación, deberá crearse en la carpeta res/layout un fichero XML con el diseño que se quiere que tenga la vista de datos (cada elemento del listado). Un ejemplo simple podría ser el siguiente (lo llamaremos *elemento.xml*):

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView android:id="@+id/miTexto1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="18sp" />
    <TextView android:id="@+id/miTexto2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="normal"
        android:textSize="12sp" />
</LinearLayout>

```

Por último se creará el adaptador, para lo cual pueden ser utilizadas dos variantes: ArrayAdapter y BaseAdapter.

6.6.1. ArrayAdapter

Este es, quizás, el adaptador más sencillo porque convierte un ArrayList de objetos en vistas, cargadas en un contenedor, que en este caso es un ListView. Este adaptador se ocupa de localizar los datos para la lista y convertirlos en un objeto para la vista.

Para ello es conveniente que se cree una clase independiente, que extenderá de ArrayAdapter, a cuyo constructor se le pasará el contexto (que suele ser la actividad principal) y la clase que se ha creado para gestionar el array de objetos con los datos que se van a mostrar.

Dentro del constructor se llamará a la superclase, pasándole como parámetros el contexto, el layout que da formato a cada elemento de la lista y los datos. Además, se pasará la referencia del objeto datos como parámetro para otros métodos (previamente lo tendremos que haber

declarado). El aspecto que tendría hasta ahora esta clase, a la que se ha llamado *Adaptador*, sería el siguiente:

```
public class Adaptador extends ArrayAdapter<Datos> {
    private Datos[] datos;
    public Adaptador(Context context, Datos[] datos) {
        super(context, R.layout.elemento, datos);
        this.datos = datos;
    }
}
```

A continuación, y dentro de esta clase, se debe sobrescribir el método (*getView*) que se ocupará de generar y llenar con los datos cada uno de los elementos del listado mostrado en la interfaz gráfica de cada elemento de la lista.

Este método será llamado cada vez que haya que mostrar un nuevo elemento de la lista y realizará el siguiente proceso. Lo primero que hará es “inflar” el layout que da formato a cada elemento (en este ejemplo, se ha llamado *elemento.XML*). Para esto, se instancia un objeto *LayoutInflater* (mostrado), se localiza su contexto (*getContext*) y se asocia la vista al elemento a “inflar”. Por último, queda tan solo referenciar los objetos que aparecen en el XML (*texto1* y *texto2*), que se llenarán con los datos que se encuentran en la posición que se va a mostrar del array.

```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater mostrado = LayoutInflater.from(getContext());
    View elemento = mostrado.inflate(R.layout.elemento, parent, false);
    TextView texto1 = (TextView) elemento.findViewById(R.id.miTexto1);
    texto1.setText(datos[position].getTexto1());
    TextView texto2 = (TextView) elemento.findViewById(R.id.miTexto2);
    texto2.setText(datos[position].getTexto2());
    return elemento;
}
```

Ya solo queda preparar la llamada desde el Java principal. Primero se prepara el array con la información para mostrar (en el ejemplo se mostraban dos cadenas de texto). Esto se puede hacer de varias maneras, una de ellas podría ser la siguiente:

```
Datos[] datos = new Datos[]{
    new Datos("Línea Superior 1", "Línea Inferior 1"),
    new Datos("Línea Superior 2", "Línea Inferior 2"),
    new Datos("Línea Superior 3", "Línea Inferior 3"),
    new Datos("Línea Superior 4", "Línea Inferior 4")};
```

Como es fácil imaginar, no es operativo incluir en la clase principal fuentes de datos con mucha información, sino que se debe recurrir a otras alternativas, como pueden ser una clase independiente, fichero en memoria, base de datos.

A continuación, se incorpora el objeto *ListView* en el XML principal de la aplicación (tal como se vio en el apartado anterior), se instancia y, por último, se crea el adaptador (que será una instancia del adaptador personalizado) y se asocia al *ListView*.

```
ListView listado = (ListView) findViewById(R.id.miLista);
Adaptador miAdaptador = new Adaptador(this, datos);
listado.setAdapter(miAdaptador);
```

Ponerle una cabecera al listado es fácil, para ello se debe crear un XML con el aspecto que se quiere que tenga la cabecera. Como ejemplo, creamos un fichero llamado *cabecera.XML* (en la carpeta layout) con una sola línea y fondo amarillo.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="25dp"
        android:text="LISTADO EJEMPLO"
        android:textStyle="bold"
        android:background="#FF200"
        android:gravity="center" />
</LinearLayout>
```

Ahora se insertará la cabecera (antes de aplicar el adaptador al listado), dos líneas de código, una que crea la vista de la cabecera y gestiona su inflado, y otra que añade la cabecera a la lista.

```
View miCabecera = getLayoutInflater().inflate(R.layout.cabecera, null);
listado.addHeaderView(miCabecera);
```

Para detectar el elemento pulsado, se utiliza un proceso similar al anteriormente visto. Se crea un escuchador al listado (*setOnItemClickListener*) para ese evento (*onItemClick*), que debe recibir cuatro parámetros, el control que contiene la lista (*AdapterView*), la vista del objeto pulsado (*View*), la posición del elemento pulsado (*position*) y el id del elemento pulsado.

Queda acceder a la vista asociada al adaptador y obtener el elemento situado en *position* usando *getItemAtPosition()* del que se puede extraer la información que interese.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> adaptador, View v, int position, long id) {
        String seleccionado = ((Datos) adaptador.getItemAtPosition(position)).getTexto();
    }
});
```

Investiga



Incorpora todos los objetos que deseas (direcciones web, CheckBox, RadioButton, imágenes...) a cada elemento de la lista. Para ello debes realizar un diseño del XML que gestiona su vista con el formato adecuado a tus necesidades y luego incorporar estos objetos al Java que gestiona los datos.

Lógicamente, la fuente de datos (en nuestro caso, el array) debe tener todos los elementos para mostrar. En caso de tratarse de imágenes, en el array debe aparecer la URL o recurso de memoria donde se encuentran.

6.6.2. BaseAdapter

Ahora se verá el uso de `BaseAdapter`, una clase muy utilizada para crear adaptadores en Android. La construcción es muy similar al de `ArrayAdapter`, ya que previamente se ha de diseñar el aspecto que se quiere que tenga cada elemento de la vista de datos, así como una clase Java que gestione los datos (pueden servir los utilizados en `ArrayAdapter`).

Este adaptador extenderá de `BaseAdapter`, que tiene una serie de métodos que es aconsejable conocer. Son estos:

- `int getCount()`
- `Object getItem(int position)`
- `long getItemId(int position)`
- `View getView(int position, View convertView, ViewGroup parent)`

De ellos, `getCount()` devuelve el número total de elementos que se mostrarán en la lista, para lo que tiene en cuenta el tamaño del array. Cuando un elemento de la lista está preparado para mostrarse, es llamado el método `getView (int i, View view, ViewGroup viewGroup)`, colaborando con la clase `LayoutInflater` para su mostrado en pantalla.

Para conocer el elemento que se encuentra en una posición específica dentro de la colección de datos, será necesario recurrir a `getItem (int i)`, mientras que `getItemId (posición int)` devuelve el id de posición del elemento.

Ahora el constructor del adaptador recibirá un `ArrayList` generado a partir del Java que gestione el suministro de datos (POJO). Y, al igual que con `ArrayAdapter`, será el método `getView` quien gestione el inflado del `ListView` con los datos aportados, obteniendo en cada caso los datos del array según la posición en la que se encuentre. El código final del adaptador quedaría de la siguiente forma:

```
public class Adaptador extends BaseAdapter {
    private ArrayList<Datos> datos;
    private Context contexto;
    public Adaptador (Context contexto, ArrayList<Datos> datos) {
        super();
        this.contexto = contexto;
        this.datos = datos;
    }
    @Override
    public View getView(int posicion, View view, ViewGroup parent) {
        LayoutInflater mostrado = LayoutInflater.from(contexto);
        View elemento = mostrado.inflate(R.layout.elemento, parent, false);
        TextView texto1 = (TextView) elemento.findViewById(R.id.miTexto1);
        texto1.setText(datos.get(posicion).getTexto1());
        TextView texto2 = (TextView) elemento.findViewById(R.id.miTexto2);
        texto2.setText(datos.get(posicion).getTexto2());
        return elemento;
    }
    @Override
    public int getCount() { return datos.size(); }
    @Override
    public Object getItem(int posicion) {return datos.get(posicion); }
    @Override
    public long getItemId(int posicion) { return posicion; }
}
```

Además, se han de hacer modificaciones en el Java principal, ya que ahora se suministrarán los datos en un `ArrayList`, quedando esa parte del código del siguiente modo:

```
ArrayList<Datos> datos = new ArrayList<Datos>();
datos.add(new Datos("Línea Superior 1", "Línea Inferior 1"));
datos.add(new Datos("Línea Superior 2", "Línea Inferior 2"));
datos.add(new Datos("Línea Superior 3", "Línea Inferior 3"));
datos.add(new Datos("Línea Superior 4", "Línea Inferior 4"));
```



Actividad propuesta 6.4

En la plataforma de Editorial Síntesis encontrarás un tutorial guiado para realizar una actividad que implemente un adaptador (BaseAdapter) en un ListView, en el que cada elemento contendrá: imagen, título, contenido y botón de radio.

Recuerda que para construir este proyecto necesitarás, además de la Activity principal (que incluirá una clase POJO para los datos), otra clase con el adaptador y dos ficheros XML: uno será la vista de la clase principal (que tendrá el ListView) y otro dará forma a los elementos del listado.

El tutorial te ayudará también a controlar los eventos de respuesta al RadioButton y al texto del Listview, tal y como aparecen en la siguiente imagen.



Terminada la actividad con ListView y adaptadores, intenta realizar las siguientes prácticas (GridView y Spinner), siguiendo un proceso muy similar al anterior.



6.7. OptionsMenu

Se trata del más simple y clásico menú, que se despliega al pulsar una tecla u opción de pantalla. Puede construirse incorporando las opciones en un fichero XML (con el nombre deseado) ubicado en res/menu, con la siguiente estructura:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/MnOp1" android:title="Opción de menú A"></item>
    <item android:id="@+id/MnOp2" android:title="Opción de menú B"></item>
    <item android:id="@+id/MnOp3" android:title="Opción de menú C"></item>
</menu>
```

Para mostrarlo (“inflarlo”) se ha de utilizar en Java el siguiente método:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.opciones, menu);
    return true;
}
```

Ya solo queda asociarle un escuchador que responda a alguno de los identificadores que se han asignado a cada opción de menú.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    String mensaje = "";
    switch (item.getItemId()) {
        case R.id.MnOp1:
            // Código para realizar
            return true;
        case R.id.MnOp2:
            // Código para realizar
            return true;
        case R.id.MnOp3:
            // Código para realizar
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Figura 6.6
Ejemplo de OptionsMenu.



Para crearlo en su totalidad desde Java, se han de introducir los identificadores y luego añadir cada una de las opciones de menú con el método OnCreateOptionsMenu.

```
private static final int MnOp1 = 1;
private static final int MnOp2 = 2;
-----
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, MnOp1, Menu.NONE, "Opción A desde Java");
    menu.add(Menu.NONE, MnOp2, Menu.NONE, "Opción B desde Java");
    return true;
}
```

En el escuchador, los casos del Switch serán 1 y 2, los valores que se le han dado a los identificadores de las opciones.

6.8. Submenú

Para crear subopciones dentro de un menú, se puede proceder igual que en el caso anterior, definiéndolo desde XML, para lo cual debe crearse un nuevo menú dentro de las etiquetas del ítem al que se quieren asociar opciones secundarias, como en el ejemplo siguiente:

```
<item
    android:id="@+id/MnOp2"
    android:title="Opción de menú B">
    <menu>
        <item android:id="@+id/MnOp2_1"
            android:title="Opción B.1" />
        <item android:id="@+id/MnOp2_2"
            android:title="Opción B.2" />
    </menu>
</item>
```

Para hacerlo desde Java, se crea una instancia de la clase SubMenú y se añade a la opción de menú de la que dependa. Luego se incorpora cada una de las opciones de submenú.

```
public boolean onCreateOptionsMenu(Menu menu) {
    SubMenu smnu = menu.addSubMenu(Menu.NONE, MnOp1, Menu.NONE, "Opción A desde Java");
    smnu.add(Menu.NONE, MnOp1_1, Menu.NONE, "Opción A.1");
    smnu.add(Menu.NONE, MnOp1_2, Menu.NONE, "Opción A.2");
    menu.add(Menu.NONE, MnOp2, Menu.NONE, "Opción B desde Java");
    return true;
}
```



Figura 6.7
Submenús en Android.



Actividad propuesta 6.5

Realiza las siguientes actividades de menú (en XML) y submenú (en Java) con escuchadores a los eventos según aparecen en las imágenes.



Opciones de menú creadas desde XML.



Pulsar Menú 1



Pulsar una opción



Pulsar una opción



Pulsar una opción



Pulsar el mes de Febrero

Opciones de menú y submenú creadas desde Java.

TEN EN CUENTA

- ✓ Los escuchadores que responden a cada uno de los eventos de las opciones de submenú se construyen de igual forma que las de las opciones principales.

6.9. Menú contextual

Son opciones de menú que aparecen en la interfaz de usuario cuando se realiza una pulsación larga sobre algún elemento de la pantalla. Para construirlo se debe asociar el menú contextual a algún objeto de pantalla, frecuentemente a una etiqueta de texto.

```
TextView etiqueta = (TextView)findViewById(R.id.texto);
registerForContextMenu(etiqueta);
```



Figura 6.8
Menús
contextuales.

Es necesario, también, definir las opciones de menú y, en su caso, submenú, que puede hacerse de cualquiera de las formas anteriores (en este ejemplo, se ha utilizado el mismo fichero XML de las veces anteriores). Luego se trabajará con una clase específica de este menú, `onCreateContextMenu`, en el que se debe construir e inflar el menú contextual.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
}
```

El escuchador para este menú se construye de manera similar a los anteriores, pero se sustituye `onOptionsItemSelected` por `onContextItemSelected`.

Recurso web

www

En este código QR encontrarás un vídeo de Codeloverlatino (Jorge Luis Villavicencio Correa), en el que seguir un tutorial para la creación de menús contextuales en Android.



6.10. Menú contextual en listas

Puede ser útil hacer que aparezcan menús contextuales en alguno de los elementos de un listado para ofrecer, por ejemplo, opciones alternativas. Siguiendo el esquema que se ha estudiado para ListView, se incluirá un objeto de lista en el fichero XML.

```
<ListView android:id="@+id/listado"
    android:layout_height="wrap_content"
    android:layout_width="match_parent" />
```

Ahora se han de crear tantos ficheros XML de menú como menús contextuales distintos se desea que aparezcan. En este caso se han definido tres (los puedes descargar de la plataforma). A manera de ejemplo, aquí se muestra uno de ellos.

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/ListaOpA1"
        android:title="OPCIÓN 1 DE MENÚ DE LISTA A"></item>
    <item android:id="@+id/ListaOpA2"
        android:title="OPCIÓN 2 DE MENÚ DE LISTA A"></item>
</menu>
```

RECUERDA

- ✓ Los identificadores de las opciones siempre deben ser distintos.

En el Java se define el array con los datos que se mostrarán, se instancia la lista, se crea el adaptador, se le asocia a la lista y se registra la lista para que tenga un menú contextual.

```
private ListView lista;
-----
String[] datos = new String[]{"OPCIÓN DE MENÚ A", "OPCIÓN DE MENÚ B", "OPCIÓN DE MENÚ C",
    "OPCIÓN DE MENÚ D", "OPCIÓN DE MENÚ E"};
lista = (ListView) findViewById(R.id.listado);
```

```
ArrayAdapter<String> adaptador = new ArrayAdapter<String>
    (this, android.R.layout.simple_list_item_1, datos);
lista.setAdapter(adaptador);
registerForContextMenu(lista);
```

Como en el apartado anterior, se ha de “inflar” el menú contextual, pero para saber cuál de ellos debemos “inflar”, obtendremos primero del adaptador el elemento que se ha pulsado.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) menuInfo;
    menu.setHeaderTitle(lista.getAdapter().getItem(info.position).toString());
    switch (info.position) {
        case 0:
            inflater.inflate(R.menu.menu_lista1, menu);
            return;
        case 1:
            inflater.inflate(R.menu.menu_lista2, menu);
            return;
        case 2:
            inflater.inflate(R.menu.menu_lista3, menu);
            return;
    }
}
```

Al igual que en el menú contextual anterior, el escuchador que se va a utilizar se construye a partir de `onContextItemSelected` y responderá una u otra cosa, según sea el identificador de la opción pulsada.



Figura 6.9
Menús contextuales en listas.

Resumen

- Android tiene variantes de menús que van evolucionando según las nuevas versiones del sistema; una de ellas es `OptionsMenu` (el más clásico se despliega al pulsar una tecla u opción de pantalla). Este, al igual que los siguientes, podemos construirlo desde XML (con las opciones en un fichero de recursos) o dinámicamente desde Java. `SubMenu`, es una variante del anterior, al que se le añaden opciones secundarias a las principales.
- También disponemos de menús contextuales, de frecuente uso y aplicables tanto a etiquetas de pantalla como a alguno de los elementos de un listado.

- Los listados son importantes objetos de Android, pertenecientes a ViewGroup, que nos muestran un conjunto de elementos ordenados (con variantes) por los que podemos navegar y seleccionar.
- Existen tres variantes principales: ListView (listado clásico), GridView (en forma de rejilla, en el que podemos configurar el número de columnas y el aspecto de estas) y Spinner (aquel cuyos elementos aparecen contraídos y solo se muestra uno antes de ser desplegado). Todos tienen sus escuchadores específicos, que nos devuelven el elemento seleccionado, la posición en la vista y en la lista.
- Los adaptadores son esenciales para permitir la inclusión de múltiples objetos en los listados, configurar el aspecto y permitir que la estética de los mismos se adapte a la requerida en la aplicación contenedora. De las variantes posibles, dos son las más usadas: los ArrayAdapter (convierte un ArrayList de objetos en vistas) y BaseAdapter (la más usada por su versatilidad y posibilidades). Tanto una como otra necesitan en su construcción un elemento (POJO) que gestione el suministro de datos.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes escuchadores utilizarías en un OptionsMenu?:
 a) onContextItemSelected.
 b) onOptionsItemSelected.
 c) onItemClickListener.
 d) onItemSelectedListener.
2. ¿Qué método se utiliza para mostrar un menú en pantalla?:
 a) onCreateOptionsMenu.
 b) getMenuInflater.
 c) registerForContextMenu.
 d) onCreateContextMenu.
3. ¿Cómo se pueden crear menús con opciones de submenús en Android?:
 a) Solo desde XML.
 b) Solo desde Java.
 c) Tanto desde XML como desde Java.
 d) Los submenús no son factibles en Android.
4. En un menú contextual de listado, ¿quién suministra la información del elemento tocado?:
 a) El propio elemento.
 b) El objeto listado.
 c) El objeto POJO.
 d) El adaptador.

5. ¿Cuál de los siguientes no es un atributo con funciones estéticas en un ListView?:
- a) divider
 - b) dividerHeight.
 - c) getItemAtPosition.
 - d) footerDividersEnabled.
6. ¿Cuál de los siguientes puede ser valor de stretchMode en un GridView?:
- a) spacingWidth.
 - b) verticalSpacing.
 - c) spacingWidthUniform.
 - d) columnWidth.
7. ¿Cómo se denominan las vistas que salen de pantalla en un listado?:
- a) ScrapViews.
 - b) Dirty View.
 - c) Recycle View.
 - d) Exit View.
8. ¿Cuál de estos elementos no se pasa como parámetro a un ArrayAdapter?:
- a) Contexto.
 - b) Layout de la clase contenedora.
 - c) Layout que da formato a los elementos.
 - d) Los datos.
9. ¿Cuál de los siguientes utilizarías como escuchador en un ArrayAdapter?:
- a) onContextItemSelected.
 - b) onOptionsItemSelected.
 - c) onItemClickListener.
 - d) onItemSelectedListener.
10. ¿Cuál de los siguientes no es un método asociado a BaseAdapter?:
- a) int getCount.
 - b) Object getItem(int position).
 - c) long getItemId(int position).
 - d) Los tres lo son.

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d

7.1. Introducción

En este capítulo se verán importantes apartados que facilitan la interrelación de la aplicación con el usuario. Se dará un primer paso en la gestión de preferencias como elemento de personalización de la APP. Por otro lado, se estudiarán toast, diálogos y notificaciones, algunas de las más frecuentes maneras que tiene un dispositivo Android de comunicarse con el usuario.

7.2. Las preferencias

Se denomina así a la información que una aplicación guarda para personalizar su interrelación con el usuario. Android da varias posibilidades para realizar dicha personalización: la clase SharedPreferences (persistencia que se estudiará en el siguiente capítulo), que almacena la información en formato clave-valor, y la clase PreferenceActivity, que se tratará en este apartado.

Esta clase ofrece una forma alternativa de definir las preferencias mediante un fichero XML, donde puede ser incluida una serie de opciones de configuración de la aplicación, creando de manera fácil las pantallas que permiten al usuario modificarlas.

El fichero XML que define las preferencias de la aplicación debe ubicarse en la ruta res/xml, donde pueden ubicarse tantos como la aplicación necesite. En el ejemplo que se expone en este apartado se ha denominado *opciones.xml*. En su interior, cada una de las alternativas debe estar incluida entre las siguientes etiquetas:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    //Preferencias definidas
</PreferenceScreen>
```

7.2.1. Categorías

En el fichero de preferencias podemos establecer diferentes secciones que se presentarán en la interfaz de usuario como apartados distintos con título propio. Esto se consigue mediante el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="CATEGORÍA 1">
        //Preferencias definidas en esta categoría
    </PreferenceCategory>
    <PreferenceCategory android:title="CATEGORÍA 2">
        //Preferencias definidas en esta categoría
    </PreferenceCategory>
</PreferenceScreen>
```

Dentro de las preferencias o de cada categoría, si así se necesita, son varias las posibles alternativas para incluir que esta clase nos ofrece, y que se ven a continuación.

7.2.2. CheckBoxPreference

Representa un tipo de opción que tan solo puede tomar dos valores distintos: activada o desactivada.

```
<CheckBoxPreference
    android:key="clave1"
    android:title="PREFERENCIA 1"
    android:summary="Descripción de la preferencia 1" />
```

Donde *title* y *summary* aparecerán como descripción en pantalla y *key* será la clave mediante la cual podremos acceder a ella para procesar su contenido.

7.2.3. EditTextPreference

Representa un tipo de opción que puede contener como valor una cadena de texto. En este caso, al pulsar sobre esta preferencia aparecerá una ventana emergente, con el título definido en *dialogTitle*, una línea de captura de texto y los botones de cancelar y aceptar.

```
<EditTextPreference
    android:key="clave2"
    android:title="PREFERENCIA 2"
    android:summary="Descripción de la preferencia 2"
    android:dialogTitle="INTRODUCE VALOR" />
```

7.2.4. ListPreference

Representa un tipo de opción que puede tomar como valor un elemento, y solo uno, seleccionado por el usuario entre una lista de valores predefinidos. Esta lista deberá definirse en el directorio res/values. El fichero que contiene la lista se ha denominado *sistemas.xml* y tiene el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="lista">
        <item>WINDOWS</item>
        <item>LINUX</item>
        <item>IOS</item>
    </string-array>
    <string-array name="clavelista">
        <item>sia1</item>
        <item>sia2</item>
        <item>sia3</item>
    </string-array>
</resources>
```

Como se puede apreciar, son dos arrays, uno denominado “lista”, que contiene las entradas (entries) que aparecerán en pantalla, y otro denominado “clavelista”, que nos definirán los índices con los que acceder a cada elemento de la lista. El resultado final de ListPreference será el siguiente:

```
<ListPreference
    android:key="clave3"
    android:title="PREFERENCIA 3"
    android:summary="Descripción de la preferencia 3"
    android:dialogTitle="ARTISTA PREFERENCIA"
    android:entries="@array/lista"
    android:entryValues="@array/clavelista" />
```

7.2.5. MultiSelectListPreference

Muy similar a ListPreference, con la diferencia de que el usuario puede seleccionar varias de las opciones de la lista de los posibles valores mostrados.

7.2.6. Acceso a las preferencias

Para mostrar las preferencias en pantalla, se debe crear una clase Java que extienda de PreferenceActivity y cuya función es la de mostrar el contenido del fichero de preferencias.

```
public class OpcionesPreferencias extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.opciones);
    }
}
```

Para “lanzar” esta clase, desde la Activity principal se puede utilizar un Intent como respuesta a cualquier evento (onClick de un botón, por ejemplo).

```
btnPreferencias.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent(Proyecto1.this, OpcionesPreferencias.class));
    }
});
```

RECUERDA

- ✓ Para que sea posible “lanzar” la actividad que gestiona las preferencias, esta debe ser incluida en el fichero del Manifest.

Para obtener las preferencias marcadas se utilizará la clase SharedPreferences, gestionada por PreferenceManager. Una instancia de esta permitirá obtener los valores de cada una de ellas a través de las claves asignadas, definiendo valores por defecto para aquellos casos que no se haya introducido ningún valor.

```

btnObtenerPreferencias.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(Proyecto1.this);
        Log.i("", "Opción 1: " + pref.getBoolean("clave1", false));
        Log.i("", "Opción 2: " + pref.getString("clave2", "No asignada"));
        Log.i("", "Opción 3: " + pref.getString("clave3", "No asignada"));
    }
});

```

Actividad propuesta 7.1



Crea una actividad que tenga en la pantalla principal dos botones: el superior, que muestre una interface de preferencias, con las opciones que se muestran en la imagen; y el inferior, que recupere los valores introducidos y los muestre mediante log en el entorno de desarrollo.



Google ha marcado esta clase como desaprobada y, aunque sigue estando plenamente operativa, recomienda que la gestión de preferencias se haga mediante la clase PreferenceFragment (que se estudiará en el apartado “Fragments” del capítulo 9) o bien mediante SharedPreferences (incluida en el capítulo 10 “Persistencia” de este libro).

7.3. Toast

Son mensajes que se muestran en pantalla durante sólo unos segundos para volver a desaparecer automáticamente, sin requerir ningún tipo de actuación. Su construcción es muy sencilla, tan solo se ha de pasar el contexto, el mensaje que se va a mostrar y un modificador que define el tiempo que estará en pantalla.



Figura 7.1
Diferentes ubicaciones del Toast.

Posteriormente, se invoca su aparición mediante la orden show. También pueden ser posicionados en cualquier lugar de la pantalla mediante setGravity.

```
Toast toast =Toast.makeText(getApplicationContext(),"MENSAJE", Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER|Gravity.LEFT,0,0);
toast.show();
```

Una alternativa opuesta a LENGTH_SHORT es LENGTH_LONG.



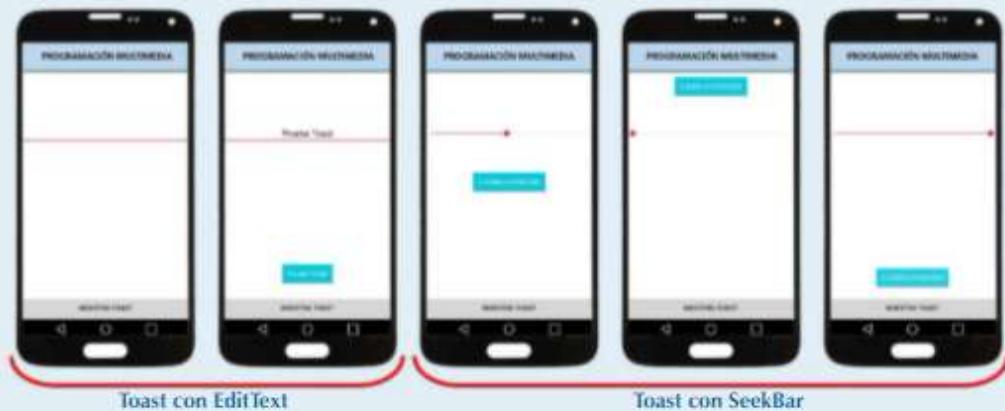
Actividad propuesta 7.2

Puedes personalizar un Toast asociándole a su vista un objeto de los que has estudiado en ViewGroup, toast.setView(layout), como puedes ver en el ejemplo que encontrarás en la plataforma de Editorial Síntesis.



Actividad propuesta 7.3

Crea las siguientes actividades basadas en Toast. La primera debe mostrar en un Toast la información introducida en un EditText. En la segunda, el Toast se mostrará más alto o más bajo en pantalla según el valor de un SeekBar. Las siguientes capturas de pantalla pueden servir como guía para la realización de esta actividad.



7.4. Diálogos

Son objetos de Android que pueden ser utilizados con distintos fines, como puede ser el mostrar un mensaje, pedir una confirmación rápida, solicitar una elección entre varias alternativas.

La forma más simple de construir un cuadro de diálogo es utilizando la subclase AlertDialog de la clase base Dialog. Esta permitirá variados diseños, teniendo en cuenta que un cuadro de diálogo consta de tres regiones, como puede verse en el siguiente esquema:

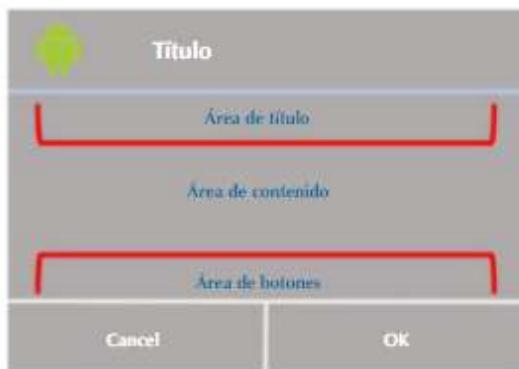


Figura 7.2
Partes de un cuadro de diálogo.

Para crear un diálogo se seguirán tres pasos: primero se instanciará mediante un constructor, luego se han de “setear” las características que se desea que tenga el cuadro de diálogo y, por último, se ha de enlazar el constructor con el objeto. Para mostrarlo se usará el comando show.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("EJEMPLO DE DIÁLOGO")
    .setTitle("TÍTULO DEL DIÁLOGO")
    .setIcon(R.mipmap.ic_launcher);
AlertDialog dialogo = builder.create();
dialogo.show();
```

Este es el diálogo básico. A continuación, se verán algunas mejoras o variantes. Se pueden añadir botones (hasta un máximo de tres) al constructor, con sus correspondientes escuchadores.

```
.setPositiveButton("PRIMERO", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Código para realizar
    }
})
.setNegativeButton("SEGUNDO", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Código para realizar
    }
})
.setNeutralButton("TERCERO", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Código para realizar
    }
})
```

Si se desea poner un listado sobre el que realizar una selección, además de definir el array que se va a mostrar, se ha de incorporar el siguiente código:

```
CharSequence[] elementos={"CASO 0","CASO 1","CASO 2","CASO 3","CASO 4","CASO 5"};
-----
.setItems(elementos, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Código para realizar
    }
});
```

TEN EN CUENTA

- ✓ El listado aparecerá en el área de contenido, por lo que no es posible mostrar simultáneamente un mensaje y una lista. También puede especificarse una lista mediante setAdapter(), lo que permite mostrar datos dinámicos (BBDD) usando un ListAdapter.



Figura 7.3
Distribución
de fragmentos
en dispositivos
de diferente
formato.

Una variante de la anterior es el incluir selección única o múltiple. En el primer caso, se utiliza la opción.setSingleChoiceItems, en lugar de.setItems. Como es de suponer, el parámetro numérico que aparece en esta opción hace referencia al elemento que aparecerá marcado por defecto a la hora de mostrarse la aplicación.

```
.setSingleChoiceItems(elementos,0, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Código para realizar
    }
});
```

En cuanto a la selección múltiple, se ha de usar una variante de escuchador.

```
.setMultiChoiceItems(elementos, null,
    new DialogInterface.OnMultiChoiceClickListener() {
        public void onClick(DialogInterface dialog, int which, boolean isChecked) {
            if (isChecked) {
                // Código para realizar
            } else {
                // Código para realizar
            }
        }
});
```

Actividad propuesta 7.4

Realiza dos actividades con cuadro de diálogo (uno con botones y otro de listado) que sean lanzados mediante un objeto Button y respondan con un Toast a la acción del usuario sobre el cuadro de diálogo. Las capturas de pantalla te servirán como guía.

**7.4.1. DatePickerDialog**

Dos cuadros de diálogo con un aspecto especial, que también puede ser interesante incluir en las aplicaciones, son el selector de fecha (Date) y el selector de tiempo (Time).

Para el selector de fecha, debe hacerse una instancia del calendario, a partir de la cual se obtendrán el día, mes y año, información que se pasará a la instancia creada del objeto DatePickerDialog. Este objeto tendrá un escuchador para el evento cambio de fecha (DateSet). Un ejemplo simple de este tipo de cuadro de diálogo puede ser el siguiente:

```
final Calendar calen = Calendar.getInstance();
int year = calen.get(Calendar.YEAR);
int month = calen.get(Calendar.MONTH);
int day = calen.get(Calendar.DAY_OF_MONTH);

DatePickerDialog fecha = new DatePickerDialog(this,
    new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int
monthOfYear, int dayOfMonth) {
            // Código para realizar
        }
    }, year, month, day);
fecha.show();
```



Figura 7.4
Ejemplo de
DatePickerDialog.

**TOMA NOTA**

También puede añadirse una barra de progreso al cuadro de diálogo, lo que suele ser muy útil en aquellas situaciones en las que el usuario debe esperar a que se realice una acción. Estos objetos se construyen como una tarea asíncrona, por lo que, para su mejor compresión, es recomendable trabajar primero este tipo de tareas. En la plataforma puedes ver un ejemplo de cuadro de diálogo con barra de progreso.

7.4.2. TimePickerDialog

El selector de hora se construye de una forma muy similar al de fecha, tan solo se han de cambiar las clases de instanciadas y el tipo de escuchadores.

```
final Calendar calen = Calendar.getInstance();
int hour = calen.get(Calendar.HOUR_OF_DAY);
int minute = calen.get(Calendar.MINUTE);

TimePickerDialog hora = new TimePickerDialog(this,
    new TimePickerDialog.OnTimeSetListener(){
        @Override
        public void onTimeSet(TimePicker view, int hourOfDay, int
        minute) {
            // Código para realizar
        }
    }, hour, minute, false);
hora.show();
```



Figura 7.5
Ejemplo
de TimePickerDialog.



Actividad propuesta 7.5

Realiza una actividad con DatePickerDialog o con TimePickerDialog, que se muestre mediante un botón y al pulsar Aceptar en el cuadro de diálogo, aparezca en un Toast la fecha u hora elegida.

7.5. Notificaciones

Una notificación es un mensaje que se puede visualizar en la parte exterior del interfaz de la aplicación que esté en ese momento en pantalla.

Es frecuente estar familiarizados con ellas, ya que son las que se muestran en el dispositivo cuando, por ejemplo, se recibe un mensaje SMS, llegan actualizaciones disponibles o se informa del reproductor de música abierto en segundo plano.

Las notificaciones pueden ser clasificadas en grupos, estilizadas mediante `setStyle()` y tienen un pequeño interfaz con el siguiente aspecto.

Figura 7.6
Elementos
de una
notificación.



Para construir una notificación se ha de seguir el siguiente proceso:

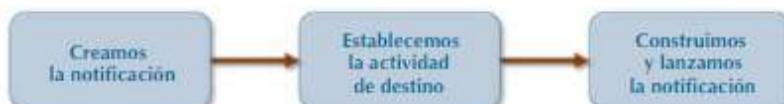


Figura 7.7
Esquema para la construcción de una notificación.

7.5.1. Crear la notificación

Como primer paso, se ha de instanciar el `NotificationManager`, que será utilizado como notificador, y se le asocia al servicio de notificaciones del dispositivo.

```

private NotificationManager notificador;
...
notificador = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
  
```

A continuación, se utiliza el constructor del `NotificationCompat`, al que se le pasa el contexto, el id del canal de notificación y todas las propiedades que se desea que tenga la notificación.

```

NotificationCompat.Builder builder =
    new NotificationCompat.Builder(this, "Canal_ID")
        .setSmallIcon(android.R.drawable.stat_sys_warning)
        .setLargeIcon(((BitmapDrawable)getResources()
            .getDrawable(R.mipmap.ic_launcher)).getBitmap())
        .setContentTitle("MENSAJE DE ALERTA")
        .setContentText("Ejemplo de notificación.")
        .setTicker("AVISO DE NOTIFICACIÓN");
  
```

Aunque es fácil imaginar el sentido de cada una de las propiedades de la notificación según su nombre, aquí aparece una pequeña descripción de cada una de ellas:

- `setSmallIcon`: ícono que aparecerá en la barra del dispositivo al llegar la notificación.
- `setTicker`: mensaje en la barra del dispositivo al llegar la notificación.
- `setLargeIcon`: ícono del interfaz de la notificación.
- `setContentTitle`: título de la notificación.
- `setContentText`: texto de la notificación.

www

Recurso web

En este vídeo de Sociedad Android podrás seguir un tutorial para la creación de notificaciones.



7.5.2. Establecer la actividad de destino

Se ha de asignar la acción mediante una instancia de la clase PendingIntent. Se puede utilizar para practicar este ejemplo cualquier Activity que ya se tenga creada o crear un simple “hola mundo”.

```
Intent intent = new Intent(this, Segunda.class);
```

Ahora se tendrá que dar una serie de pasos para crear la pila de trabajo y que el dispositivo interprete el proceso de ejecución cuando “salte” la notificación. Primero se deberá registrar en el Manifest la actividad que va a responder y designarle la actividad padre, que será aquella que la va a lanzar.

```
<activity android:name=".Segunda"
    android:parentActivityName=".Notificacion">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="es.curso.Notificacion"/>
</activity>
```

A continuación, se inicializará la pila de retroceso (BackStack), se añadirá la actividad padre a la pila y se asociará el Intent creado a la pila.

```
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(Notificacion.class);
stackBuilder.addNextIntent(intent);
```

Por último, se ha obtener y asignar el PendingIntent.

```
PendingIntent resultPendingIntent = stackBuilder.getPendingIntent (0,
PendingIntent.FLAG_UPDATE_CURRENT);
builder.setContentIntent(resultPendingIntent);
```

7.5.3. Construir y lanzar la notificación

Se vuelve a trabajar con el builder, donde se configura la forma de cancelación y se asocia el builder al notificador, también se le asocia un identificador y se lanza la notificación.

```
builder.setAutoCancel(true);
notificador.notify(1, builder.build());
```



Figura 7.8
Ejemplo de notificación.

Si ahora se ejecuta la aplicación, se observará que en el ícono de la misma aparecerá un aviso, y si desplaza hacia abajo la barra superior del dispositivo, aparecerá el interfaz de la notificación con un aspecto muy similar al de la figura 7.8.

Pulsando sobre este se te abrirá la actividad que tenías asociada al Intent. Si se quiere que se abra directamente esta segunda actividad sin que tenga que intervenir el usuario, tan solo se ha de añadir, antes de la línea de lanzamiento, el siguiente código:

```
builder.setFullScreenIntent(resultPendingIntent, true);
```

También pueden añadirse parámetros de seguridad. Si se desea que se muestre visible con el aspecto que he visto hasta ahora, se ha de escribir:

```
builder.setVisibility(NotificationCompat.VISIBILITY_PUBLIC);
```

Otras alternativas son VISIBILITY_PRIVATE, que muestra información mínima de la notificación, como el ícono y el título, pero nada sobre el detalle en sí. O en caso de que se desee que no muestre ninguna información de la notificación, se utilizará VISIBILITY_SECRET.

Para cancelar mediante código la notificación, se puede invocar:

- `notificador.cancel(1);` Cancelaría la notificación identificada con ese número.
- `notificador.cancelAll();` Cancelaría todas las notificaciones.

Actividad propuesta 7.6



Realiza una actividad que sea lanzada mediante un botón, cuya actividad destino sea un Intent (ligado a PendingIntent), del tipo implícito que abra el navegador con la página de Google.



PARA SABER MÁS

Android Studio dispone de una herramienta que te permitirá generar con facilidad los iconos para las notificaciones, *Configure Image Asset*.

Figura 7.9
Herramienta Configure Image Asset de Android Studio.



Resumen

- Las características y propiedades de una aplicación pueden ser gestionadas mediante las clases PreferenceActivity, PreferenceFragment y SharedPreference.
- Android dispone de variadas formas de mandar mensajes e interactuar con el usuario. Las más simples son los Toast (similares a los Snackbar en Material Design). Otros son los variados cuadros de diálogo y las comunicaciones, objetos que nos permiten lanzar actividades remotas.
- Los Toast son mensajes personalizables que se muestran en pantalla durante unos segundos para luego volver a desaparecer automáticamente, sin requerir ningún tipo de actuación.
- Los diálogos nos permiten gran versatilidad, ya que pueden solo mostrar un mensaje, o pedir una confirmación rápida, solicitar una elección entre varias alternativas... Además, entre este grupo podemos incluir los DatePickerDialog y los TimePickerDialog, que pueden dotar de gran profesionalidad a nuestro interface.
- Algo más complejos en su construcción son las notificaciones, muy frecuentes en las aplicaciones actuales (SMS, WhatsApp, Email...). Son mensajes que se pueden visualizar en la parte exterior del interfaz de la aplicación sin interferir en la aplicación que tenemos en primer plano en ese momento. Tienen variados atributos de configuración y nos permiten lanzar actividades secundarias mediante PendingIntent.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes objetos de comunicación ha aparecido con Material Design?:
 a) Toast.
 b) Snackbar.
 c) Diálogos.
 d) Notificaciones.
2. Con respecto a los Toast, ¿cuál de estas afirmaciones es incorrecta?:
 a) Se muestran en pantalla durante unos segundos.
 b) Para su desaparición, debes tocar la pantalla con el dedo.
 c) Puedes configurar su duración.
 d) Puedes configurar su aspecto.
3. ¿Cuáles de las siguientes áreas no pertenecen a un cuadro de diálogo?:
 a) Área de título.
 b) Área de contenido.
 c) Área de listado.
 d) Área de botones.

4. ¿Cuál de los siguientes modificadores utilizarías en un cuadro de diálogo con listado?:
- a) setItems.
 - b) setSingleChoiceItems.
 - c) setMultiChoiceItems.
 - d) Cualquiera de los anteriores.
5. ¿Cuál de esta información podemos obtener de un TimePickerDialog?:
- a) Día.
 - b) Hora.
 - c) Segundos.
 - d) Año.
6. ¿Quién gestiona la aparición de una notificación?:
- a) El NotificationManager.
 - b) El servicio de notificaciones del dispositivo.
 - c) El NotificationCompact.
 - d) La propia aplicación.
7. ¿Cuál de los siguientes atributos no son aplicables a una notificación?:
- a) setSmallIcon.
 - b) setTicker.
 - c) setTimer.
 - d) setContentTitle.
8. ¿Cuál de las siguientes acciones hay que seguir para crear la pila de trabajo en una notificación para llamar a la actividad destino?:
- a) Crear una instancia de la clase PendingIntent.
 - b) Instanciar el TaskStackBuilder para inicializar la pila de retroceso.
 - c) Asociar el Intent a la pila de retroceso.
 - d) Todas las anteriores son necesarias.
9. ¿Cómo podemos lanzar la segunda actividad en una notificación sin la intervención del usuario?:
- a) Mediante builder.setFullScreenIntent(resultPendingIntent, true).
 - b) Mediante builder.setAutoCancel(true).
 - c) Mediante stackBuilder.addParentStack(Notificación.class).
 - d) Mediante builder.setContentIntent(resultPendingIntent).
10. ¿Cuál de estas no son opciones de visibilidad de una notificación?:
- a) VISIBILITY_PUBLIC.
 - b) VISIBILITY_PRIVATE.
 - c) VISIBILITY_HIDDEN.
 - d) VISIBILITY_SECRET.

SOLUCIONES:

1. a b c d
 2. a b c d
 3. a b c d
 4. a b c d

5. a b c d
 6. a b c d
 7. a b c d
 8. a b c d

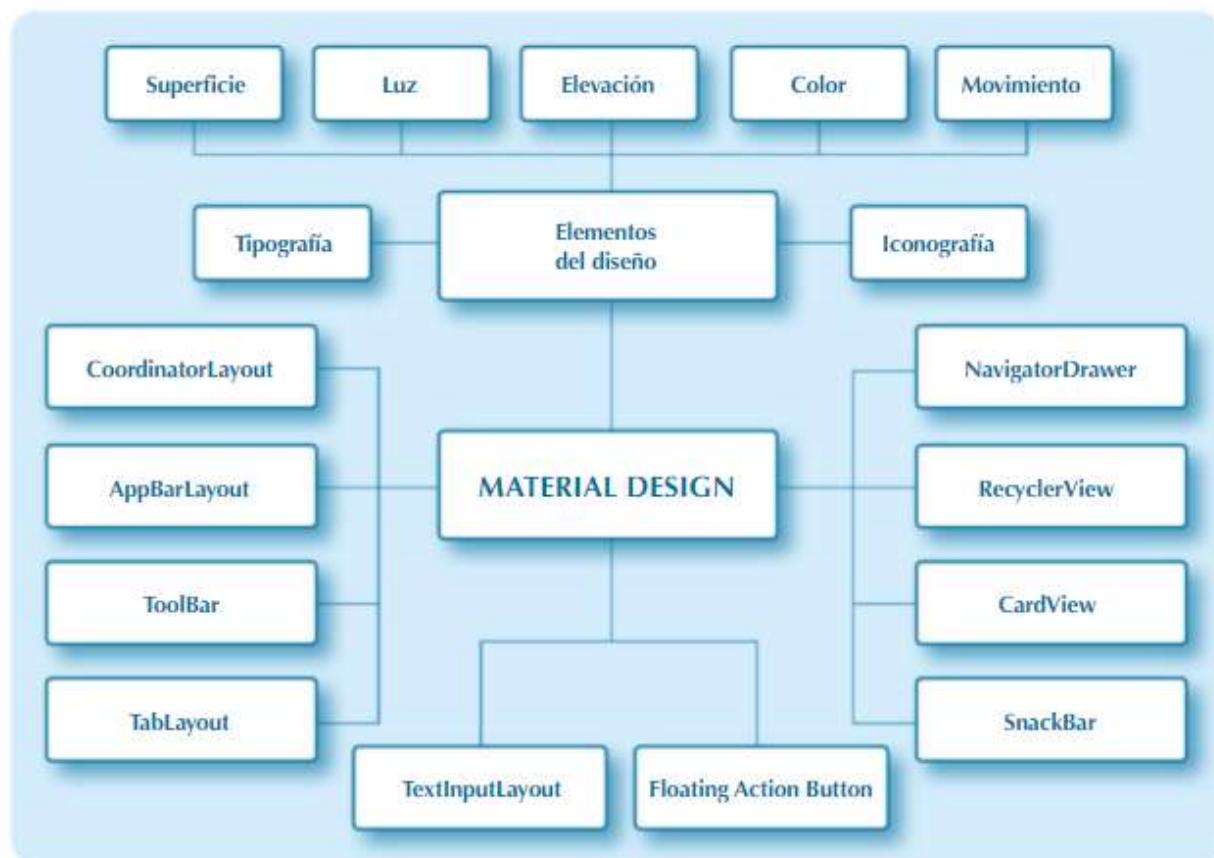
9. a b c d
 10. a b c d

Material Design

Objetivos

- ✓ Familiarizarse con los principios de diseño de Material Design.
- ✓ Usar con asiduidad los recursos de color, dimensiones, estilo...
- ✓ Utilizar la luz y el color como parte importante en el diseño de las aplicaciones.
- ✓ Seguir pautas estandarizadas para estilizar las actividades.
- ✓ Manejar con destreza la configuración del tema en actividades y aplicaciones.
- ✓ Conocer y aplicar las nuevas librerías de objetos que aporta Material Design.
- ✓ Construir aplicaciones con los nuevos objetos de Material Design.
- ✓ Dar respuesta a los eventos que ocurrán a los objetos de las nuevas librerías.

Mapa conceptual



Glosario

AppTheme.NoActionBar. Tema que suele incluir Android para desactivar la ActionBar.

CoordinatorLayout. Nuevo tipo de layout usado como contenedor principal, que va a controlar la animación de los elementos de la interfaz.

Estilo. Colección de propiedades que especifican la apariencia y el formato de una vista.

Palette. Librería que permite extraer colores de una imagen.

Pattern. Clase que deja verificar que los caracteres introducidos en un campo están permitidos y que la longitud de la cadena es la adecuada.

Ripple. Efecto visual comparable al de la expansión de una onda en una superficie líquida.

Scrapped. Se denominan así a las vistas no mostradas y que están marcadas para su destrucción o reutilización.

Tema. Estilo que se da a toda una aplicación o una actividad en concreto.

8.1. Introducción

Material Design es el diseño oficial de Android planteado no solo para dispositivos móviles, sino para todo tipo de contenidos digitales. Fue presentado durante el Google I/O 2014, bajo la autoría del chileno Matías Duarte e implementado a partir de la versión 5.0 de Android (API 21).

Incluye consejos y también imágenes el diseño, basadas en colores y formas que representan la propia física de la naturaleza, para intentar conseguir aplicaciones que resulten atractivas, que faciliten la usabilidad y favorezcan la empatía de los usuarios.

Es un concepto, una filosofía, unas pautas enfocadas al diseño utilizado en Android, pero también en la web y en cualquier plataforma. Se trata de animaciones lógicas con piezas colocadas en un espacio (lugar) y con un tiempo (movimiento) determinado, que se aproximan a la realidad, guiándose por las leyes de la física.

Recursos web



A través de estos códigos QR encontrarás interesantes vídeos sobre Material Design (Google Design) y los elementos del diseño (Google Developer).



1



2

8.2. Elementos del diseño

Los principales elementos que se han de tener en cuenta en el diseño son los que se desarrollan a continuación.

8.2.1. La superficie y la luz

En este nuevo concepto el material está tratado como una metáfora, las superficies dan pistas visuales basadas en la realidad, creando nuevas propuestas alternativas a las impuestas por el mundo físico. Se utilizan los principios de la superficie y la luz para transmitir el estado de los objetos y su interrelación.

8.2.2. Elevación

Uso de sombras atrevidas, luces y colores vibrantes para mostrar qué superficie está delante de otras, lo que permite ver al usuario la importancia de cada elemento, centrar su atención y establecer jerarquías. Si un elemento tiene sombra significa que está por encima del resto y posiblemente sea más importante.



Figura 8.1
La elevación
con Material
Design.

8.2.3. Colores

El uso del color debe ser uno de los elementos básicos en el diseño de los elementos visuales de nuestras aplicaciones. Los colores deben ser intensos para enfatizar los elementos importantes de la interface, se deben asignar colores primarios para los más destacados. El color debe seguir los principios de ser jerárquico (marca la importancia), legible (que no sea confuso sobre su fondo) y expresivo (que refuerce la marca).



SABÍAS QUE...

Palette es una librería que permite extraer colores de una imagen. La extracción de colores puede ser una operación lenta, por lo que se recomienda no usarlo en el hilo principal, sino que se utilicen para ello métodos asíncronos. Debe incluirse en la compilación:

```
dependencies { compile 'com.android.support:palette-v7:22.2.0' }
```

Un ejemplo de aplicación podría ser el siguiente:

```
Palette.generateAsync(bitmap, new Palette.PaletteAsyncListener() {
    @Override
    public void onGenerated(Palette palette) {
        titulo.setTextColor(palette.getLightVibrantColor(defaultTextColor));
        autor.setTextColor(palette.getVibrantColor(defaultTextColor));
    }
});
```

8.2.4. Movimiento

El movimiento refuerza el papel del usuario, sin que interfiera su labor sobre la aplicación, existiendo coherencia entre las animaciones y la esencia de la misma. Animaciones y transiciones son significativas, ofrecen continuidad visual entre una pantalla y la siguiente. Debe estar regido por los principios de la física de la realidad, objetos dotados de velocidad y de una dirección que nos indican si es más o menos ligero, flexible o de dónde proviene.



PARA SABER MÁS

Lollipop, siguiendo con los principios de Material Design, incorpora un feedback visual al usuario, que le informa que su acción está en marcha. Esto se consigue mediante el efecto Ripple, comparable al de la expansión de una onda, desde el punto tocado, en una superficie líquida.

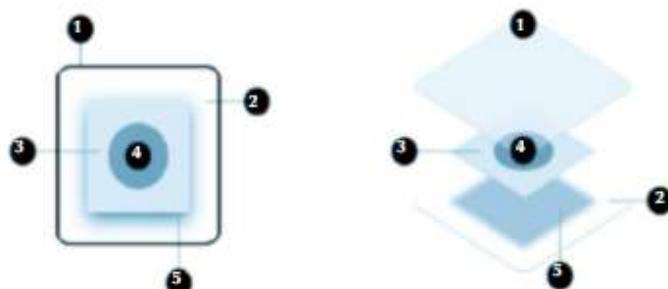
8.2.5. Tipografías

Se hace uso de fuentes sencillas para presentar diseño y contenido de manera más clara y eficiente. La escala de tipo incluye una gama de estilos variados, algunos en mayor tamaño, para hacerlos vistosos a pesar de ser simples.

8.2.6. Iconografía

La variedad de iconos se multiplica a la vez que lo hace la variedad de diseños posibles. Se busca que los iconos transmitan la idea central de una manera simple y amigable. Se alinean al contenido según una cuadrícula base de 8 dp para mantener la coherencia visual. Los botones siguen una jerarquía, en la que el botón flotante (FAB) es el de mayor importancia.

Figura 8.2
Elementos
de los iconos
en Material Design
(1. Terminar, 2. Fondo,
3. Primer plano,
4. Color, 5. Sombra).



RECUERDA

- ✓ Como ya se vio, un *estilo* en Android es una colección de propiedades que especifican la apariencia y el formato de una View. Un *tema* es un estilo que se da a toda una aplicación o una actividad en concreto, pero no a una View individual. Cuando se pone un estilo como tema, todas las vistas de la actividad o aplicación usarán las propiedades de este estilo. Todo ello viene regulado en el Manifest, que aplica en orden jerárquico un tema (`android:theme`) a la aplicación o actividades en ella existentes, haciendo referencia a uno o varios de los estilos definidos en el fichero styles dentro de values. Con Material Design son muchas las posibilidades que se ofrecen al definir estilos propios y temas (especialmente en aspectos de color), por lo que para sacarle el máximo partido a la funcionalidad de estas nuevas librerías es aconsejable acostumbrarse a definir las propiedades estéticas (estilo) de los objetos que vayan a incluirse en las aplicaciones.

8.3. Trabajar con Material Design

Para poder desarrollar, incluyendo las nuevas funcionalidades de Material Design, el entorno (Android Studio) debe reconocerlas, y para ello es necesario que en el Grandle esté implementada la librería que provee de dicha funcionalidad.

Esto se puede hacer de dos maneras, la primera y más artesana, editando el fichero build.gradle (Module: app), que se encuentra dentro de la carpeta Greadle Scripts, e incluyendo esta implementación a mano. La línea que se ha de añadir será la siguiente: `implementation 'com.android.support:design:28.0.0'`, teniendo en cuenta que la versión de esta (28.0.0) debe coincidir con la de la librería `appcompat-v7` que se tenga instalada. En la figura 8.3 se muestra una posible captura de las dependencias del build.gradle:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

Figura 8.3
Contenido de parte
del archivo
build.gradle
(Module app).

La segunda posibilidad, más automática, consiste en seleccionar, a la hora de crear un nuevo proyecto, alguna plantilla que incluya objetos Material Design, como puede ser Basic Activity. Si lo practicas, observarás que el resultado es el mismo.

8.4. Layouts y ToolBar

Para estudiar este apartado se ha de crear un nuevo proyecto para la API 21, utilizando como plantilla Basic Activity y denominando la actividad principal *Actividad1*. Como se ha visto en el apartado anterior, ya aparecerán implementadas las librerías de Material Design.

Si se ejecuta la aplicación, el aspecto que tendrá será el mostrado. El conocido “Hola Mundo”, al que se añade la entrada a un OptionsMenu (arriba a la derecha) y un ícono circular (abajo a la derecha). A este nuevo ícono ya nos hemos referido con el nombre de Floating Action Button (FAB), que suele aparecer en más de una de las aplicaciones de uso frecuente en plataformas móviles, y en una de las nuevas y más llamativas librerías incorporadas con Material Design.

Ahora se trabajará con los ficheros de la vista para entender un poco la construcción de aplicaciones con estas nuevas funcionalidades. Primero se puede observar que, en esta ocasión, han aparecido dos ficheros XML en la carpeta layout de los recursos: `activity_actividad1.xml` y `content_actividad1.xml`.

El segundo es similar al que aparece en la vista de una actividad que se crea con la plantilla Empty Activity, con un ConstraintLayout como contenedor y una etiqueta de texto (TextView) de contenido.

Donde aparece una importante diferencia es en el fichero `activity_actividad1.xml`, que tiene una estructura anidada similar a la siguiente:



Figura 8.4
Actividad con FAB.

```
<android.support.design.widget.CoordinatorLayout>
    <android.support.design.widget.AppBarLayout>
        <android.support.v7.widget.Toolbar/>
    </android.support.design.widget.AppBarLayout>
    <include layout="@layout/content_actividad1" />
    <android.support.design.widget.FloatingActionButton/>
</android.support.design.widget.CoordinatorLayout>
```

En la capa externa, se encuentra un CoordinatorLayout como contenedor principal, un nuevo tipo de layout, que va a controlar la animación de algunos de los nuevos elementos de la interfaz. Aunque al principio dé la sensación de que todo sigue igual, se apreciará su utilidad al añadir elementos, que se comportarán de manera más dinámica en sus transiciones y movimientos.

Dentro de este, dos elementos aportados por Material Design: por un lado, otro contenedor, AppBarLayout, similar a un LinearLayout vertical utilizado para lograr diversos comportamientos de desplazamiento, como colapso, espacio flexible y retorno rápido. Dentro del cual podemos incluir objetos como ActionBar o ToolBar (como en este caso).

El otro objeto que también aparece dentro del CoordinatorLayout es el Floating Action Button, del que ya se ha hablado. Y como importante línea de código, un *include*, que enlaza con el otro fichero XML de la vista, actuando, a todos los efectos, como si el código del otro fichero estuviera allí escrito.

8.4.1. ToolBar

Desde la salida de Android 3.0 (nivel de API 11), se incorporó al interface, en la parte superior del mismo, una barra denominada ActionBar. Esta podría acoger elementos que servían de enlace a una serie de utilidades de nuestra aplicación: icono, botones de acción, view control, menú...

ActionBar ha ido evolucionando y adquiriendo nuevas funciones, lo que ha hecho que variara su aspecto según la versión de Android instalada en el dispositivo. Material Design incorpora en su biblioteca un widget (android.support.v7.widget.Toolbar) con similar funcionalidad, la clase ToolBar, más flexible y personalizable. Google recomienda que esta última sustituya a ActionBar en sus aplicaciones.

Recurso web

Puedes acceder a la página sobre ToolBar en Material Design con este código QR.



Para construirla, se incorporará el widget en el fichero de la vista (XML), dentro de un contenedor (recomendable CoordinatorLayout y AppBarLayout), con identificador, atributos de posicionamiento y estética (elevación, background, theme...), como podría ser:

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:elevation="8dp"
    android:background="#3d9cdc"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

Si se “lanza” la aplicación, aparecerá la barra, pero sin contenido (figura 8.5, imagen 1). Esto es debido a que aún no está instanciada en Java, lo cual se hará de la siguiente manera:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```



TOMA NOTA

Android añade, a la hora de definir un tema, una ActionBar entre las propiedades del mismo; en las actividades realizadas hasta ahora nos aparecía por defecto esta barra en la parte superior de la pantalla. Esta es incompatible con la implementación de una ToolBar, por lo que si se observa en el Manifest (al crear la aplicación con una plantilla de Material Design), se configura un tema con un estilo, generalmente AppTheme.NoActionBar, definido en res/styles.xml, que tiene desactivada la ActionBar (mediante el comando item name="windowActionBar">false). Esto es importante ya que, si se añade ToolBar a una actividad con la ActionBar activada, esta aplicación no se lanzará.

Ya con la barra instanciada en Java (figura 8.5, imagen 2), se pueden añadir recursos a la misma, como un menú (siguiendo los pasos vistos en el capítulo “Listados y Menús”), que nos podrían dejar la barra con el aspecto de la imagen 3 (figura 8.5). Si se desea que alguna de las opciones aparezca directamente en la ToolBar (figura 8.5, imagen 4), se ha de codificar en el fichero res/menu, de la siguiente manera:

```
<item
    android:id="@+id/barra_nuevo"
    android:title="ANADIR"
    app:showAsAction="always" />
```

Por último, para que alguna de estas opciones aparezca como ícono (figura 8.5, imagen 5), el código sería:

```
<item
    android:id="@+id/barra_editar"
    android:icon="@drawable/editar"
    android:title="EDITAR"
    app:showAsAction="always" />
```



Figura 8.5
Ejemplos de variantes de Toolbar.



Actividad propuesta 8.1

En la plataforma de Editorial Síntesis dispones de un tutorial para realizar una aplicación práctica con Toolbar. Sigue los pasos para desarrollarla.

8.5. TabLayout

Otra de las mejoras, tanto en diseño como en construcción, que se nos provee desde Lollipop, son las nuevas pestañas (tabs), componente importante de la interfaz que permite dividir y organizar los objetos que deben aparecer en pantalla, facilitando la navegación. Para ello, se ha de añadir este widget a la vista (recomendable debajo del Toolbar) de la siguiente forma:

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabs"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:tabMode="fixed"
    app:tabGravity="fill"/>
```

Como opciones de app:tabMode se dispone de fixed y scrollable, que alinea las tabs a la izquierda, y app:tabGravity, que permite llenar todo el ancho de la pantalla con las tabs o agruparlas en el centro de la misma (center). Otros atributos, entre los numerosos de este objeto, son:

- app:tabIndicatorColor: establece el color del indicador de la pestaña.
- app:tabIndicatorHeight: establece la altura del indicador de la pestaña seleccionada.
- app:tabSelectedTextColor: establece los colores del texto para los diferentes estados.

Con esto se tiene configurado el contenedor para las tabs, pero aún no aparece ninguna. Para ello, se ha de instanciar el TabLayout en Java y, posteriormente, añadir las tabs necesarias (figura 8.6, imagen 1).

```
TabLayout tabs = (TabLayout) findViewById(R.id.tabs);
tabs.addTab(tabs.newTab().setText("PESTAÑA 1"));
tabs.addTab(tabs.newTab().setText("PESTAÑA 2"));
tabs.addTab(tabs.newTab().setText("PESTAÑA 3"));
```

Si se desea crear las tabs directamente con un ícono (figura 8.6, imagen 2), la forma de añadir las sería la siguiente (se pueden incluir íconos propios en la carpeta Drawable):

```
tabs.addTab(tabs.newTab().setIcon(android.R.drawable.ic_media_play));
tabs.addTab(tabs.newTab().setIcon(android.R.drawable.ic_media_pause));
tabs.addTab(tabs.newTab().setIcon(android.R.drawable.ic_media_previous));
```

Cada vez que se crea una tab, el sistema le añade directamente un índice para que después sea posible acceder a ellas, manipularlas y controlar los eventos. De tal manera, que para añadir íconos a unas pestañas creadas con textos, se ha de codificar lo siguiente (figura 8.6, imagen 3).

```
tabs.getTabAt(0).setIcon(android.R.drawable.ic_media_play);
tabs.getTabAt(1).setIcon(android.R.drawable.ic_media_pause);
tabs.getTabAt(2).setIcon(android.R.drawable.ic_media_previous);
```

Para controlar los eventos se utiliza el escuchador addOnTabSelectedListener, que permite controlar tres eventos, pestaña seleccionada, deselegionada y reselecciónada (figura 8.6, imagen 4).

```
tabs.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        String mensaje="Tab Seleccionada: número "+tab.getPosition();
        Toast toast =Toast.makeText(getApplicationContext(),mensaje, Toast.LENGTH_SHORT);
        toast.show();
    }
    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
        //Código para ejecutar
    }
    @Override
    public void onTabReselected(TabLayout.Tab tab) {
        //Código para ejecutar
    }
});
```



Figura 8.6
Ejemplos
de variantes
de tabs.

Habitualmente, el cambio de tab implica un cambio total del contenido de la pantalla (a excepción de las barras de aplicaciones), por lo que es frecuente, casi necesario, el utilizar con ellas los fragmentos (se estudiará en el capítulo siguiente). De una manera simple, se podrían definir como actividades incrustadas dentro de otras. Un cambio de tab supondría un cambio de fragmento.

Recurso web

Puedes acceder a la web sobre TabLayout en Material Design con el siguiente código QR.



Otra funcionalidad importante que suele añadirse a las tabs es la de utilizar con ellas un paginador (ViewPager). Es posible que por el número de pestañas se pueda superar el ancho de la pantalla; este permitirá pasar de unas a otras, desplazándolas con el dedo.

```
<android.support.v4.view.ViewPager
    android:id="@+id/paginador"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

**Actividad propuesta 8.2**

Para hacer operativa esta funcionalidad, debes utilizar fragmentos y un adaptador para poder incluirlos en la aplicación (que herede de FragmentStatePagerAdapter).

En la plataforma de Editorial Síntesis encontrarás un tutorial para realizar una actividad completa con tabs y fragmentos.

**8.6. TextInputLayout**

Uno de los nuevos componentes en Material Design, que pretende estilizar los campos de texto con animaciones, etiquetas flotantes y asignación de mensajes de error a la hora de validar el contenido.

Recurso web

Puedes acceder a la web sobre TextInputLayout en Material Design con este código QR.



Presenta una animación del texto auxiliar en la parte superior para crear un comportamiento práctico entre el significado del campo de texto y su contenido. Ubica la etiqueta de los campos de texto dentro del mismo (inline) para, posteriormente, mostrarlas en la parte superior del campo mientras este tenga el foco o se esté introduciendo algún valor. De igual modo, puede mostrar un contador con los caracteres introducidos y el número máximo se caracteriza por los siguientes:

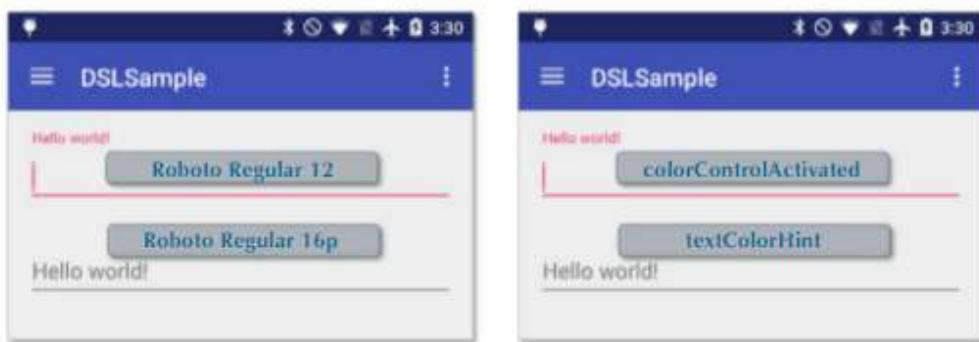


Figura 8.7
Fuentes
y colores
utilizados en
TextInputLayout.

Para construirlo, se trabaja con la vista, donde se anida un EditText en un TextInputLayout. En este ejemplo, se puede ver el aspecto que tendría antes y después de coger el foco.

```
<android.support.design.widget.TextInputLayout
    android:id="@+id/control_nombre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:counterEnabled="true"
    app:counterMaxLength="30"
    app:errorEnabled="true">

    <EditText
        android:id="@+id/campo_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nombre"
        android:inputType="text"
        android:singleLine="true" />
</android.support.design.widget.TextInputLayout>
```



Figura 8.8
Ejemplo de
TextInputLayout.

Como se puede deducir, este ejemplo tiene activados los atributos de mostrar el contador y detectar errores, y tiene marcada una longitud máxima de treinta caracteres.

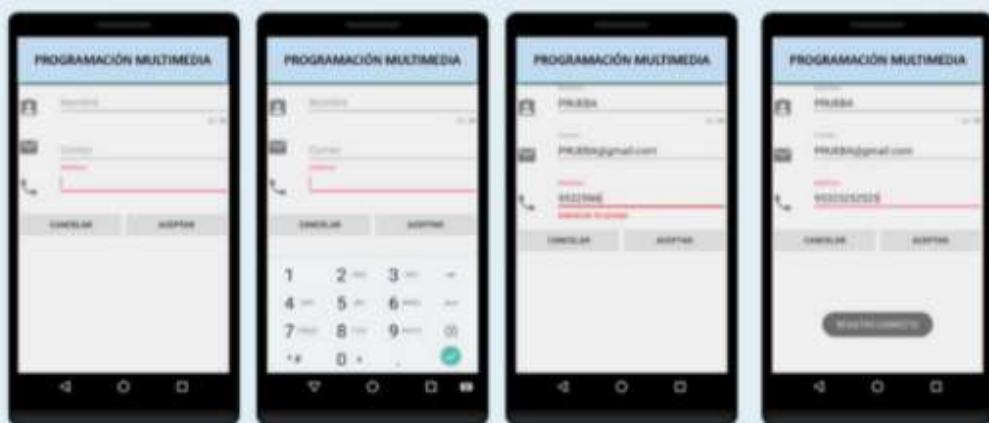
En cuanto al escuchador, para este objeto se puede utilizar el addTextChangedListener para el EditText, al que pasaremos como parámetro un TextWatcher(), que nos sobrescribe tres métodos:

- beforeTextChanged. Responde cuando toma el foco y antes de que el texto cambie.
- onTextChanged. Responde cuando cambia el texto.
- afterTextChanged. Responde cuando deja el foco tras cambiar el texto.



Actividad propuesta 8.3

En la plataforma de Editorial Síntesis encontrarás un tutorial para realizar una actividad con este objeto, que además de utilizar varios de sus atributos, realiza una validación de los datos para varios tipos de EditText.



Para validar los datos, suele utilizarse un patrón (Pattern), que verifica que los caracteres introducidos están permitidos y que la longitud de la cadena es la adecuada. Un ejemplo sería el siguiente:

```
Pattern patron = Pattern.compile("[a-zA-Z ]+\\$");
if (!patron.matcher(nombre).matches() || nombre.length() > 30) {
    controlNombre.setError("Error en Nombre");
    return false;
} else {
    controlNombre.setError(null);
}
return true;
```

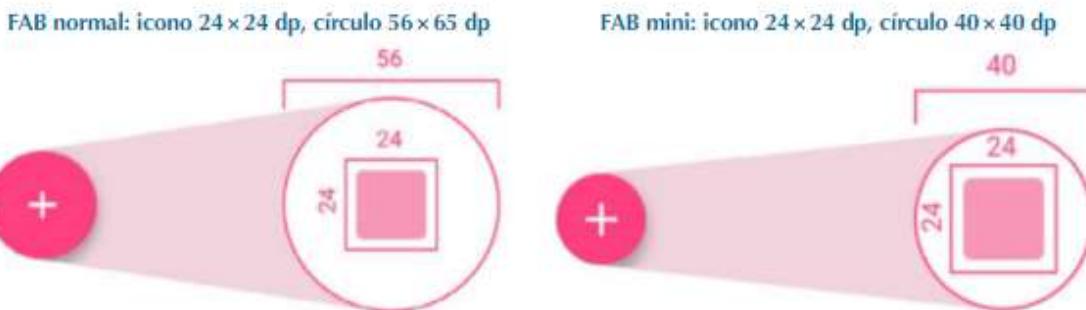
8.7. Floating Action Button (FAB)

Una de las aportaciones más populares de Material Design es el botón de acción flotante, que da acceso rápido a acciones comunes o importantes de una aplicación. Aparece sobre el contenido de la pantalla, generalmente de forma circular y con un ícono en su interior. Los FAB nos aparecen en tres tipos diferentes: normal, mini y extendido.

Recurso web

Puedes acceder a la página sobre FAB en Material Design con este código QR.



**Figura 8.9**

Tipos de Floating Action Buttons.

Para incluir un FAB en nuestra aplicación (existe una plantilla que ya lo hace), se añade el siguiente código al fichero XML de la actividad:

```
    android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        app:srcCompat="@android:drawable/ic_dialog_email" />
```

El manejador de eventos para este objeto es similar a los trabajados anteriormente.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Código para realizar
    }
});
```



Investiga

Una de las posibilidades que debemos usar con este objeto son las animaciones, para su aparición, desaparición o despliegue. Esto hará nuestras aplicaciones mucho más atractivas. Para ello igual, que ya vimos con los objetos básicos de Android, tan solo se ha de definir la animación en un fichero XML en el directorio res/anim, cargar esta animación, abrir = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.abrir), y luego aplicársela al FAB, fab.startAnimation(abrir).

En la plataforma de Editorial Síntesis dispones de un ejemplo con varios Floating Action Button animados. Investiga y práctica.

8.8. Snackbar

Similar a un Toast. Muestra información sobre algún evento u operación realizada por el usuario mediante un mensaje corto (solo uno en cada momento), que aparece desde el inferior de la pantalla por un corto periodo de tiempo y desaparece de manera automática (aunque también puede ser descartado por el usuario).

Recurso web

Puedes acceder a la página web sobre Snackbar en Material Design con este código QR.



La construcción es sencilla y muy parecida al Toast, donde se le pasa al constructor la vista (contexto), el texto que se va a mostrar y la duración. Posteriormente, se invoca el mostrado.

```
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "TEXTO PARA MOSTRAR", Snackbar.LENGTH_LONG).show();
    }
});
```

Las posibilidades de duración son:

- Snackbar.LENGTH_SHORT -> 1500 milisegundos.
- Snackbar.LENGTH_LONG -> 2750 milisegundos.
- Snackbar.LENGTH_INDEFINITE.

También puede personalizarse el tiempo mediante.setDuration(Tiempo en Milisegundos);

```
Snackbar.make(view, "TEXTO PARA MOSTRAR", Snackbar.LENGTH_LONG).setDuration(300).show();
```

Por otro lado, si se pone un tiempo largo o indefinido, se puede cancelar su presentación mediante el comando.dismiss(). Además, este objeto puede hacerse interactivo añadiendo una etiqueta de texto que responda al pulsado (onClick).

```
Snackbar miSnackbar = Snackbar.make(view, "TEXTO A MOSTRAR", Snackbar.LENGTH_LONG)
    .setAction("PULSE AQUÍ", new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            //Código para realizar
        }
    });
miSnackbar.show();
```



Investiga

Utiliza también findViewById para poner la vista del Snackbar de la siguiente forma:

```
Snackbar.make(findViewById(R.id.contenedor), "TEXTO", Snackbar.LENGTH_LONG).show();
```

La vista del Snackbar también puede ser personalizada. Para ello tan solo debe instanciarse y trabajar el texto que en ella aparece.

```
// Cambia el color del texto interactivo
snackbar.setActionTextColor(Color.BLUE);

// Cambia el color del texto del mensaje
View miVistaSb = snackbar.getView();
TextView textoSb = (TextView) miVistaSb.findViewById(android.support.design.R.id.snackbar_text);
textoSb.setTextColor(Color.RED);
```

TEN EN CUENTA

- ✓ Para que este objeto pueda ser ocultado mediante desplazamiento con el dedo, solo hay que incluirlo dentro de un CoordinatorLayout.

Otra funcionalidad interesante que tiene este objeto es poder añadir un callback a los eventos de aparición y desaparición.

```
.addCallback(new Snackbar.Callback() {
    public void onShown(Snackbar snackbar) {
        //Código para realizar
    }
    public void onDismissed(Snackbar snackbar, int event) {
        //Código para realizar
    }
});
```

8.9. CardView y RecyclerView

CardView es otro de los llamativos objetos incorporados con Material Design, que permite mostrar información (texto, imágenes, enlaces...) dentro de una tarjeta (card). Es frecuente, aunque no obligatorio, su uso asociado al nuevo contenedor RecyclerView.

WWW

Recurso web

Puedes acceder a la web sobre CardView en Material Design con este código QR.



Google nos recomienda una serie de principios en el uso de estos objetos: que la tarjeta sea identificable como una sola unidad, que no dependa de los elementos circundantes y que no pueda fusionarse con otra o dividirse en varias. También, en la página enlazada en el recurso web, nos aconseja lo que se debe hacer o no con CardView.

Su construcción es sumamente sencilla, ya que solo se trata de incluir en el XML un contenedor de tipo widget.CardView con los objetos que se desean que aparezcan en la tarjeta (CardView).

```
<android.support.V7.widget.CardView
    card_view:cardCornerRadius="4dp"
    card_view:cardElevation="4dp"
    card_view:cardUseCompatPadding="true">
    <RelativeLayout
        ...
    </RelativeLayout>
</android.support.V7.widget.CardView>
```

Son los atributos del CardView los que darán vistosidad y marcarán la diferencia con respecto a un contenedor de los hasta ahora utilizados. Los de más frecuente uso son los siguientes:

- cardBackgroundColor: color de fondo de la tarjeta.
- cardCornerRadius: define el radio de la esquina de la tarjeta.
- cardElevation: elevación de la tarjeta.
- cardMaxElevation: elevación máxima.
- cardPreventCornerOverlap: habilita/deshabilita el relleno del contenido para evitar recortes.
- cardUseCompatPadding: agrega espacio alrededor de la vista de la tarjeta.
- contentPadding: agrega el relleno entre la tarjeta y el contenido de la tarjeta.



Figura 8.10
Elementos de un CardView.



Actividad propuesta 8.4

Realiza tres actividades que muestren composiciones con CardView lo más similares a las capturas mostradas en la imagen. Puedes hacerlas con las imágenes que deseas (las que aparecen las encontrarás en la plataforma de Editorial Síntesis).



8.9.1. RecyclerView

Es una nueva clase que hereda de ViewGroup, que permite mostrar grandes colecciones o conjuntos de datos. Solo se dedica a reciclar vistas, otros componentes serán responsables de acceder a los datos y mostrarlos.

Para su construcción, necesita codificarse tanto la vista (XML) como el Java. En el primer caso, el más simple, se ha de incluir el widget de la siguiente forma:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/reciclador"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="3dp"
    android:scrollbars="vertical"
    android:background="@color/colorDivider"/>
```

Al igual que se vio con las clases ListView, GridView, para poder incluir en cada uno de los elementos de la lista que se va a reciclar objetos variados y poder configurar su aspecto, será necesaria la ayuda de un adaptador.

En este caso, el adaptador deberá extender de RecyclerView.Adapter, y su constructor recibirá como parámetro la lista fuente de datos, que le será suministrada por una clase POJO, como ya se vio anteriormente (en nuestro caso, se llama Encapsulador).

```
public class Adaptador extends RecyclerView.Adapter<Adaptador.AnimeViewHolder> {
    private List<Encapsulador> items;
    public Adaptador(List<Encapsulador> items) { this.items = items; }

    public static class AnimeViewHolder extends RecyclerView.ViewHolder {
        //Declaración de objetos

        public AnimeViewHolder(View vista) {
            super(vista);
            //Enlace de los objetos con sus vistas
        }
    }

    @Override
    public int getItemCount() {return items.size();}

    @Override
    public AnimeViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.carta, viewGroup, false);
        return new AnimeViewHolder(v);
    }

    @Override
    public void onBindViewHolder(AnimeViewHolder viewHolder, final int i) {
        //Seteado" de objetos para ser mostrados
    }
}
```

Además, el adaptador consta de un método getItemCount, otro que crea la vista del contenedor (onCreateViewHolder) y un tercero que se usa para enlazar las vistas (onBindViewHolder).

Para crear la vista, el adaptador necesita que se le suministre el layout que será inflado (en este caso, carta.xml, un CardView similar al visto anteriormente) y que mostrará la información recibida de la fuente de datos.

Además, en el Java, RecyclerView necesita del método setLayoutManager(LayoutManager layout) para asignar un tipo de layout que le indique cómo debe renderizar la vista (la forma en las que se mostrarán los CardView). Nos brinda un nuevo componente, de nombre Recycler, una clase encargada de gestionar si una vista será desechara o separada (scrapped o detached) para su reutilización.



Figura 8.11
Funcionamiento
del LayoutManager.

Si una vista es *scrapped* indica que continúa enlazada con el RecyclerView, pero se ha marcado para su posible eliminación o reutilización. Cuando es reutilizada, es considerada *dirty* y el adaptador es quien debe volver a enlazarla antes de ser mostrada.

En el proceso de reutilización, una vista se puede encontrar en dos momentos. *Scrap Heap* es cuando aún se puede interactuar con las vistas que, sin estar siendo utilizadas, van a volver a utilizarse de forma inmediata. Para ello se dispone del método detachAndScrapView(View child, RecyclerView recycler).

Recycler Pool son aquellas vistas que no se necesitarán más, pudiéndose utilizar el método removeAndRecycleView(View child, RecyclerView recycler) para removerlas definitivamente.



SABÍAS QUE...

Dependiendo del tipo de LayoutManager utilizado, los CardView pueden mostrarse en la pantalla de nuestro dispositivo de diferente manera. Cada uno de estos tipos tiene unos atributos de configuración, como se puede ver en la página oficial de Android.

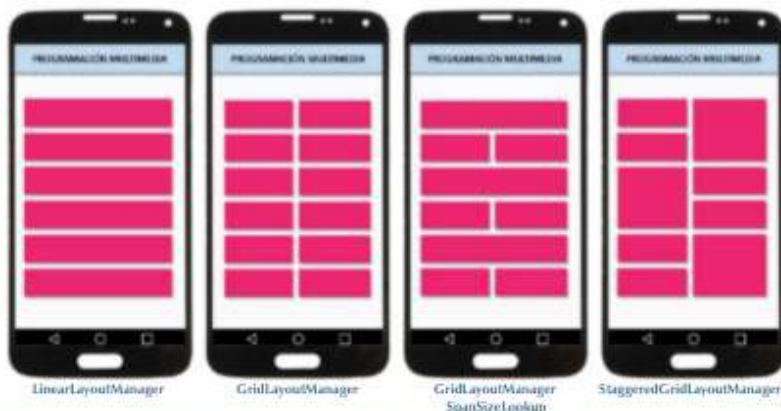


Figura 8.12
Variantes
de LayoutManager.

Ya con todo definido, en Java se han de declarar los tres elementos de la arquitectura:

```
private RecyclerView reciclar;
private RecyclerView.Adapter adaptador;
private RecyclerView.LayoutManager layManager;
```

En el siguiente paso, se instanciarán el reciclar (asociarlo a la vista y modificar alguno de sus atributos) y el LayoutManager, que se asociará al RecyclerView.

```
reciclar = (RecyclerView) findViewById(R.id.reciclar);
reciclar.setHasFixedSize(true);
layManager = new LinearLayoutManager(this);
reciclar.setLayoutManager(lManager);
```

Se instanciará el adaptador, con la clase que gestiona los datos como parámetro, y se asociará al reciclar.

```
adaptador = new Adaptador(datos);
reciclar.setAdapter(adaptador);
```

Los datos provendrán, en este caso, de un ArrayList, que será rellenado con ayuda del POJO (Encapsulador).

Queda añadir un manejador de eventos para las tarjetas y los objetos tocados (del desplazamiento de los CardView en pantalla se ocupa el RecyclerView). Acceder dentro del RecyclerView no es tarea fácil, ya que no dispone de escuchador propio. Se puede hacer de varias maneras, algunas de ellas escuchando desde el adaptador y otras desde la clase principal.

Quizás la más simple sea con el Java del adaptador, dentro del onBindViewHolder, ya que el acceso al RecyclerView no es tarea fácil. Para ello, se ha de añadir un listener al objeto que se va a escuchar.

```
viewHolder.objeto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Código para realizar
    }
});
```

Para identificar y obtener alguno de los campos del elemento de la lista tocado, en el onClick se obtendrá el contexto de la lista y localizará el campo según el índice que proporciona el Holder.

```
view.getContext()
items.get(i).get_campoBuscado()
```

También se puede escuchar el Holder directamente, implementando un listener en el método AnimeViewHolder y sobrescribiendo onClick. La posición tocada se obtendrá del adaptador.

```

public static class AnimeViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    //Declaración de objetos
    public AnimeViewHolder(View vista) {
        super(vista);
        //Enlace de los objetos con sus vistas
        vista.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        int posicion = this.getAdapterPosition();
        //Código para realizar
    }
}

```

Por último, y quizás la manera más profesional, sería trabajar con el Java principal, añadiendo en el onCreate un TouchListener al reciclador. En su interior, se incorporan tres métodos.

- onInterceptTouchEvent.
- onTouchEvent.
- onRequestDisallowInterceptTouchEvent.

Para que funcione correctamente, antes se instanciará el detector de eventos GestureDetector. En el interior de onInterceptTouchEvent se detectará e instanciará el elemento tocado. Si no es nulo, se localizará el elemento en la tabla y mostrará el campo deseado del mismo.

```

reciclador.addOnItemTouchListener(new RecyclerView.OnItemTouchListener() {
    GestureDetector gesDete = new GestureDetector(getApplicationContext(),
        new GestureDetector.SimpleOnGestureListener() {
            @Override public boolean onSingleTapUp(MotionEvent e) {
                return true;
            }
            @Override
            public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {
                View tocado = rv.findChildViewUnder(e.getX(), e.getY());
                if (tocado!= null && gestureDetector.onTouchEvent(e)) {
                    int posicion = rv.getChildAdapterPosition(tocado);
                }
                return false;
            }
            @Override
            public void onTouchEvent(RecyclerView rv, MotionEvent e) { }
            @Override
            public void onRequestDisallowInterceptTouchEvent(boolean disallowIntercept) { }
        });
});

```



Actividad propuesta 8.5

En la plataforma de Editorial Síntesis dispone de un tutorial con RecyclerView. Sigue los pasos para realizar una aplicación completa.

8.10. Navigator Drawer

Es un nuevo interfaz de usuario en forma de panel lateral que puede contener el menú principal u otras opciones de navegación de la aplicación, y que permanece oculto por defecto. Para mostrarse, se puede tocar el ícono de la aplicación o deslizar el dedo desde el lateral izquierdo de la pantalla. Para cerrarlo, igualmente, hay que tocar el ícono de la aplicación, deslizar el panel hacia el lado izquierdo, tocar el contenido de la aplicación fuera del Navigator o presionar el botón de atrás del móvil.

 WWW

Recurso web

Puedes acceder a la web sobre Navigation Drawer en Material Design con este código QR.





PARA SABER MÁS

El uso de Navigation Drawer nos permite cambiar el paradigma de navegación entre pantallas de nuestra aplicación, pudiendo hacer navegación profunda, cruzada o en árbol.



Figura 8.13
Paradigmas de navegación en una aplicación.

Los elementos base en la construcción de un Navigation Drawer están formados por los títulos, subtítulos, divisores, contadores, iconos y elementos plegables. Al igual que otros objetos llegados con Material Design, se suelen utilizar recursos propios de color, dimensiones, estilos...

Para su construcción, se utiliza en el XML un `DrawerLayout`, un contenedor especial de la librería de soporte que alberga dos tipos de vistas, la que conforma la actividad principal (enlazada mediante un `include`) y la que da formato visual al `Navigation Drawer`.

```
<android.support.v4.widget.DrawerLayout
    <include layout="@layout/contenido"/>
    <android.support.design.widget.NavigationView
        android:id="@+id/navigation"
        app:headerLayout="@layout/cabecera"
        app:menu="@menu/menu_navegacion" />
</android.support.v4.widget.DrawerLayout>
```

Esta segunda (un widget, como en otros casos) tiene, a su vez, dos elementos principales, el HeaderLayout, para el diseño de la cabecera, y el menú, que conforma el aspecto y opciones del menú que será desplegado con el Navigation Drawer. Una interesante opción de configuración es activar que este panel lateral ocupe toda la pantalla, lo que se consigue mediante el atributo android:fitsSystemWindows="true".

Para poder trabajar con este objeto Java, se ha de declarar, referenciar y, además, crearle un escuchador.

```
private DrawerLayout drawerLayout;
drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);

private void setupDrawerContent(NavigationView navigationView) {
    navigationView.setNavigationItemSelectedListener(
        new NavigationView.OnNavigationItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(MenuItem menuItem) {
                menuItem.setChecked(true);
                //Oódigo para realizar
                return true;
            }
        });
}
```

Al igual que se hizo con los tabs, este objeto se suele utilizar con fragmentos que van cambiando el aspecto y contenido de la pantalla principal de la aplicación, según la selección realizada.



Actividad propuesta 8.6

En la plataforma de Editorial Síntesis dispones de un tutorial para realizar una aplicación con Navigation Drawer, con fragmentos y escuchadores para los elementos que la componen.



Resumen

- Material Design es un concepto, una filosofía, son pautas enfocadas al diseño oficial de Android, que fue presentado durante el Google I/O 2014 e implementado a partir de la versión 5.0 de Android.
- Son elementos claves del diseño: la gestión de la luz, la superficie, la elevación, el color, el movimiento, las tipografías apropiadas y el uso coherente de la iconografía.
- AppBarLayout es un nuevo contenedor utilizado para lograr diversos comportamientos de desplazamiento, como colapso, espacio flexible y retorno rápido.
- Material Design incorpora en su biblioteca ToolBar, un widget con similar funcionalidad a ActionBar, pero con importantes mejoras en diseño y funcionamiento sobre este.
- TabLayout facilita la construcción de aplicaciones utilizando pestañas (tabs) para la navegación entre actividades (fragmentos), además de mejoras en el estilizado de las mismas, forma de mostrarse. Permite utilizar ViewPager para realizar el cambio, desplazándolas con el dedo.
- Una mejora sobre EditText nos la da TextInputLayout, que además de estilizar los campos de texto con animaciones, etiquetas flotantes, permite mostrar mensajes de error a la hora de validar el contenido de un campo de edición de texto.
- Una de las aportaciones más populares de Material Design es el botón de acción flotante (FAB), que da acceso rápido a acciones comunes o importantes de una aplicación.
- Un Snackbar nos muestra un mensaje corto que aparece desde el inferior de la pantalla, desapareciendo de manera automática o por intervención del usuario.
- CardView es otro de los objetos incorporados con Material Design, con numerosos atributos que permite mostrar información (texto, imágenes, enlaces) dentro de una tarjeta. Un grupo de estos objetos suelen mostrarse dentro de un RecyclerView, una nueva clase que hereda de ViewGroup, que permite mostrar grandes colecciones o conjuntos de datos.
- Un novedoso objeto es Navigator Drawer, un interfaz de usuario en forma de panel lateral que puede contener el menú principal u otras opciones de navegación de la aplicación.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿A partir de qué versión de Android apareció Material Design?:

- a) Versión 4.1 Jelly Bean.
- b) Version 4.4 KitKat.
- c) Versión 5.0 Lollipop.
- d) Versión 6.0 Marshmallow.

2. ¿Cómo se llama el efecto visual comparable a una onda sobre la zona pulsada?:

- a) Palette.
- b) Swapp.
- c) Fade.
- d) Ripple.

3. ¿Qué fichero contiene las dependencias a las librerías de Material Design?:

- a) Manifest.
- b) build.gradle.
- c) generatedJava.
- d) setting.gradle.

4. ¿Cómo se denomina el layout que Material Design propone como contenedor principal?:

- a) AppBarLayout.
- b) TabLayout.
- c) TextInputLayout.
- d) CoordinatorLayout.

5. ¿Qué tema debes utilizar en tu actividad para poder usar ToolBar?:

- a) AppTheme.ActionBar.
- b) Cualquiera que ponga windowActionBar en true.
- c) Cualquiera que ponga windowActionBar en false.
- d) No hay que tocar los temas para este objeto.

6. ¿Cuál de los siguientes es un método de addTextChangedListener?:

- a) beforeTextFinish.
- b) onTextChanged.
- c) afterTextFinish.
- d) Todos los son.

7. ¿Cuál de los siguientes no es un tamaño del Floating Action Button?:

- a) Normal.
- b) Mini.
- c) Extendido.
- d) Ampliado.

8. ¿Cuál es la duración de Snackbar.LENGTH_LONG?:

- a) 1500 milisegundos.
- b) 2500 milisegundos
- c) 2750 milisegundos.
- d) 3750 milisegundos.

9. ¿Qué atributo agrega espacio alrededor de un CardView?:

- a) cardCornerRadius.
- b) cardPreventCornerOverlap.
- c) contentPadding.
- d) cardUseCompatPadding.

10. ¿Cuál de los siguientes no es un tipo de LayoutManager?:

- a) LinearLayoutManager.
- b) GridLayoutManager.
- c) SpanSizeLayoutManager.
- d) StaggeredGridLayoutManager.

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d

Persistencia

Objetivos

- ✓ Valorar la importancia de la perdurabilidad de la información en las aplicaciones.
- ✓ Incluir la posibilidad de conservar las preferencias del usuario.
- ✓ Conocer la utilidad de mantener la experiencia del usuario en las aplicaciones.
- ✓ Gestionar los recursos de almacenamiento, respetando la capacidad del dispositivo.
- ✓ Hacer un uso coherente de la memoria interna como almacenamiento.
- ✓ Utilizar con asiduidad la tarjeta de memoria como recurso de almacenamiento.
- ✓ Practicar con las clases más importantes gestoras de persistencia.
- ✓ Adquirir destreza en la gestión y mantenimiento de bases de datos.
- ✓ Utilizar los servicios de almacenamiento en la Red.
- ✓ Hacer transmisiones entrantes y salientes, tanto HTTP como HTTPS.
- ✓ Construir y dar operatividad a clientes y servidores por Sockets.
- ✓ Conocer la utilidad de los gestores de contenidos.
- ✓ Crear proveedores de contenidos propios.
- ✓ Acceder a proveedores de contenidos públicos.

Mapa conceptual



Glosario

Environment Class. Clase pública de Android que da acceso a las variables del entorno.

Cursor. En bases de datos, este término hace referencia a una estructura de control utilizada para el recorrido de los registros obtenidos como resultado de una consulta.

Débilmente tipado. Se utiliza esta expresión para hacer referencia a aquel sistema que permite utilizar datos de un tipo como si fueran de otro tipo distinto. Pudiendo, por ejemplo, dar un valor entero a una variable que anteriormente contenía una variable de cadena.

Sockets. Definido como punto final de una conexión, es un método para la comunicación entre software cliente y software servidor, utilizando una red.

SQLite. Sistema de dominio público de gestión de bases de datos relacional, escrito en C e incluido en una biblioteca relativamente pequeña.

Tarjeta de memoria montada. Aquella que se encuentra reconocida por el dispositivo y disponible para ser usada.

TLS. Heredero de SSL, es un protocolo criptográfico, que proporciona seguridad en las comunicaciones por red.

UTF-8 (8-bit Unicode Transformation Format). Es un formato de codificación de caracteres Unicode e ISO 10646 que utiliza símbolos de longitud variable.

10.1. Introducción

Con el término *persistencia* nos referimos, en relación con una aplicación, a la capacidad que debe tener esta para que los datos por ella manipulados perduren en el tiempo tras la ejecución de la misma. Básicamente, consiste en almacenar la información en un medio secundario, no volátil, para su posterior recuperación y utilización, independiente en el tiempo del proceso que los creó.

Android ofrece varias opciones para asegurar los datos de una aplicación, utilizándose una u otra variante dependiendo de las necesidades específicas de la misma. Este sistema permite desde almacenar y compartir las preferencias de las aplicaciones, hasta realizar almacenamientos remotos utilizando recursos y protocolos de red, pasando por persistencia local, en ficheros, tarjetas de almacenamiento y bases de datos relacionales, entre otros. Información (especialmente la que se encuentra almacenada en estas últimas) que es posible poner a disposición del sistema y de los usuarios que hagan uso del mismo, mediante los proveedores de contenidos.

10.2. Preferencias compartidas

La clase `SharedPreferences` permite guardar y recuperar en formato de pares de clave-valor información como booleanos, elementos flotantes, valores enteros, valores largos y strings. Es decir, cada una de ellas estará compuesta por un identificador único y un valor asociado a dicho identificador. Estos datos se conservarán de una sesión a otra, incluso si la aplicación finaliza.

RECUERDA

- ✓ Como ya se ha visto en capítulos anteriores, de manera similar a lo tratado en este apartado, Android dispone de la clase `PreferenceActivity`, que puede ser usada para conservar las preferencias de usuario, de manera automática y con una interfaz fácil de construir.

Para dar funcionalidad a esta clase, se dispone de dos métodos:

- `getSharedPreferences()`: cuando se necesitan varios archivos de preferencias.
- `getPreferences()`: si solo es necesario un archivo de preferencias para la actividad.

Su construcción es sencilla, instanciamos la clase, poniendo el nombre tan solo en caso de utilizar varios archivos y el modo de acceso.

```
SharedPreferences prefs = getSharedPreferences(Nombre_Archivo, Context.MODE_PRIVATE);  
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
```

Los modos posibles de acceso son los siguientes:

- MODE_PRIVATE. Solo la aplicación creadora tiene acceso a estas preferencias.
- MODE_WORLD_READABLE. Todas las aplicaciones pueden leerlas, pero solo la creadora puede modificarlas.
- MODE_WORLD_WRITEABLE. Todas las aplicaciones pueden leer y modificar estas preferencias.



SABÍAS QUE...

Los MODE_WORLD_READABLE y MODE_WORLD_WRITEABLE han quedado en desuso, ya que a partir de Android 7.0 (API nivel 24), Android lanza si se usa una Security Exception.

Para escribir valores, se debe instanciar el editor sobre la clase de preferencias creada, se añaden los valores mediante putString(), putBoolean(),.putInt(), putFloat() y se guarda con commit().

```
SharedPreferences.Editor editor = prefs.edit();
editor.putString("Clave1", "Valor1");
editor.putString("Clave2", "Valor2");
editor.commit();
```

Estas preferencias se almacenarían en un fichero XML, ubicado dentro del dispositivo móvil en una ruta con el siguiente patrón:

```
/data/data/paquete.java/shared_prefs/nombre_archivo.xml
```



Investiga

Siguiendo la opción de menú en Android Studio, View>Tool Windows>Device File Explorer, podemos acceder al explorador de archivos en el dispositivo móvil. En la ruta antes mostrada, podemos encontrar el archivo de preferencias de la actividad realizada. En este caso, ubicación y contenido serían los siguientes (el archivo se denomina VerAndroid). Investiga y practica.

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<map>
    <string name="nombre">D00001</string>
    <string name="lanzada">2016</string>
</map>
```

Figura 10.1

Captura de la ubicación y contenido de un fichero de preferencias.

Para recuperar los valores, se utilizarán métodos como `getBoolean()`, `getString()`...

```
String valor1 = prefs.getString("Clave1", "Valor por Defecto 1");
String valor2 = prefs.getString("Clave2", "Valor por Defecto 2");
String valor3 = prefs.getString("Clave3", "Valor por Defecto 3");
```

El primer parámetro hace referencia a la clave, mientras que el segundo es el que tomaría esa clave, en caso de no encontrar ninguno asignado o que esta ni siquiera existiese.



Actividad propuesta 10.1

Realiza una aplicación con una actividad que contenga dos objetos `EditText`, dos botones (Guardar y Recuperar) y dos objetos `TextView`. Al pulsar el botón Guardar, el contenido de los `EditText` será almacenado mediante `SharedPreferences` y al pulsar el botón Recuperar, el contenido almacenado será mostrado en los dos objetos `TextView`.

10.3. Almacenamiento en la memoria interna del dispositivo

Con los inconvenientes que puede imponernos lo limitado de la memoria del dispositivo, este suele ser un medio bastante usado cuando el volumen de datos que se va a almacenar no es demasiado alto; además, no es necesario solicitar permisos en el Manifest.

Para el proceso de escritura, Android nos proporciona el método `OutputStreamWriter`, en colaboración con el método `openFileOutput`, al que tenemos que suministrar los parámetros: Nombre de Archivo y Modo de Acceso.

- `MODE_PRIVATE`. Solo accesible por la aplicación creada (crea o sobrescribe si ya existe).
- `MODE_APPEND`. Añade contenido a un archivo existente en el dispositivo.
- `MODE_WORLD_READABLE`. Permite la lectura por otras aplicaciones (en desuso desde la API 17).
- `MODE_WORLD_WRITEABLE`. Permite la escritura por otras aplicaciones (en desuso desde la API 17).

Para evitar excepciones, utilizamos en su construcción el bloque `try/catch` de la siguiente manera:

```
try {
    OutputStreamWriter miFichero = new OutputStreamWriter(openFileOutput("fichero.txt", Context.MODE_PRIVATE));
    miFichero.write("TEXTO DE PRUEBA");
    miFichero.close();
} catch (Exception ex) {
    //Código ejecutado
}
```

Este fichero es almacenado en el dispositivo en: data/data/paquete.java/files/fichero.txt

Para leer los datos almacenados, también con try/catch, se usa el método BufferedReader, abriéndolo previamente con el método openFileInput.

```
try {
    BufferedReader miFichero = new BufferedReader(new InputStreamReader(openFileInput("fichero.txt")));
    String texto = miFichero.readLine();
    miFichero.close();
} catch (Exception ex) {
    //Código ejecutado
}
```

Actividad propuesta 10.2



Realiza una aplicación con una actividad que contenga un objeto EditText, dos botones (Guardar y Recuperar) y un objeto TextView. Al pulsar el botón Guardar, el contenido del EditText será almacenado en la memoria interna y al pulsar el botón Recuperar, el contenido del fichero será mostrado en el objeto TextView.

10.4. Lectura de un recurso de la aplicación

Es una variante del anterior, que solo se ha de utilizar cuando no es necesario realizar modificaciones sobre los ficheros, al estar limitado a solo lectura. Por tanto, previamente se ubicará el fichero (en formato TXT), con la información que se vaya a necesitar, dentro de la carpeta /res/raw de la aplicación.

```
try {
    InputStream fichraw = getResources().openRawResource(R.raw.fichero);
    BufferedReader miFichero = new BufferedReader(new InputStreamReader(fichraw));
    String linea = miFichero.readLine();
    fichraw.close();
} catch (Exception ex) {
    //Código ejecutado
}
```

10.5. Persistencia en la tarjeta de almacenamiento externo

Una interesante posibilidad es conservar la información en una tarjeta de almacenamiento externo (microSD), especialmente cuando se desea guardar alto volumen de datos. Si bien hay que tener en cuenta que los archivos creados por este método son accesibles para todos los usuarios, por lo que no es un método recomendable para almacenar información sensible.

Este método necesita solicitar los siguientes permisos en el Manifest:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

El primer paso recomendado a la hora de trabajar con ficheros en memoria externa es asegurarnos de que dicha memoria está presente y disponible para leer o escribir en ella. Esto se puede hacer mediante el método Environment.getExternalStorageState().

```
boolean sdExiste = false;
boolean sdEscritura = false;
String sdEstado = Environment.getExternalStorageState();
if (sdEstado.equals(Environment.MEDIA_MOUNTED))
{
    sdExiste = true;
    sdEscritura = true;
}
else if (sdEstado.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
{
    sdExiste = true;
    sdEscritura = false;
}
else
{
    sdExiste = false;
    sdEscritura = false;
}
```

Algunos de los posibles estados de la tarjeta son:

- Environment.MEDIA_MOUNTED. Disponible, “montada” para lectura y escritura.
- Environment.MEDIA_MOUNTED_READ_ONLY. Disponible, “montada” para solo lectura.
- Environment.MEDIA_UNMOUNTED. Disponible, pero “no montada”.
- Environment.MEDIA_REMOVED. No disponible.

Una vez se ha comprobado que la tarjeta de memoria existe y está montada, procedemos a escribir en ella, en el directorio raíz o en un directorio definido, para lo que se establece la ruta y se crea el fichero de texto destino. Todo ello dentro de un bloque try/catch.

```
try
{
    //Ruta para directorio raiz
    File ruta_sd = Environment.getExternalStorageDirectory();
    //Ruta para directorios predefinidos
    File ruta_sd = Environment.getExternalStorageDir(null);
    File mifichero = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
    OutputStreamWriter ficht =
        new OutputStreamWriter(
            new FileOutputStream(mifichero));
    ficht.write("TEXTO DE PRUEBA.");
    ficht.close();
}
catch (Exception ex)
{ Log.e("Ficheros", "Error al escribir fichero en la SD");}
```

Si se define como null el parámetro de getExternalFilesDir, el fichero de almacenamiento será creado dentro de la siguiente ruta: <raíz_mem_ext>/Android/data/paquete.java/files. Teniendo como alternativas los siguientes parámetros:

- DIRECTORY_MUSIC.
- DIRECTORY_PICTURES.

- DIRECTORY_MOVIES.
- DIRECTORY_RINGTONES.
- DIRECTORY_ALARMS.
- DIRECTORY_NOTIFICATIONS.
- DIRECTORY_PODCASTS.

Para leer los datos almacenados, una vez seleccionada la ruta, tan solo se ha de cambiar el sentido del buffer.

```
try
{
    //Ruta para directorio raíz
    File ruta_sd = Environment.getExternalStorageDirectory();
    //Ruta para directorios predefinidos
    File ruta_sd = Environment.getExternalStorageDir(null);
    File mifichero = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
    BufferedReader fichsd = new BufferedReader(
        new InputStreamReader(
            new FileInputStream(mifichero)));
    String texto = fichsd.readLine();
    fichsd.close();
}
catch (Exception ex)
{ Log.e("Ficheros", "Error al escribir fichero en la SD");}
```

Actividad propuesta 10.3



Realiza una aplicación con una actividad que contenga un objeto EditText, dos botones (Guardar y Recuperar) y un objeto TextView. Al pulsar el botón Guardar, el contenido del EditText será almacenado en la tarjeta SD y al pulsar el botón Recuperar, el contenido del fichero será mostrado en el objeto TextView.

10.6. Persistencia en bases de datos (SQLite)

Cuando el volumen de información dificulta el realizar su salvaguardada por alguno de los métodos anteriormente relacionados y deseamos asegurar la información en el propio dispositivo, utilizar bases de datos puede ser una buena alternativa.

De las opciones posibles que las compatibilidades de Android ofrecen, SQLite es un motor de bases de datos de pequeño tamaño, no necesita servidor, precisa poca configuración, es transaccional y, por supuesto, de código libre. Guarda toda la base de datos en un único fichero y no requerirá la instalación adicional de ningún gestor de bases de datos. Como inconveniente de este método, es que no mantiene la integridad de datos, le falta soporte Unicode y una interfaz gráfica de administración.

Recurso web**www**

ORMLite es un paquete que se implementa en la base de datos y permite hacer llamadas directas a la API para acceder a SQLite. En su página web puedes consultar sus posibilidades.



La base de datos se almacenará en /data/data/paquete.java/databases/nombre_bbdd/. Por defecto, todas las bases de datos son privadas y solo accesibles por la aplicación creadora.

Para gestionar la base de datos, deberemos usar la clase SQLiteOpenHelper, para lo que se debe crear una clase Java que herede de esta y cuyo constructor reciba como parámetros, el contexto, el nombre de la base de datos, el CursorFactory (suele ser null para utilizar el estándar de SQLiteCursor) y la versión del esquema de base de datos (para posibles migraciones).

Esta clase sobrescribirá el método onCreate, que es llamado cuando la base de datos se crea por primera vez. Es donde se define la estructura de las tablas y pueden cargarse los datos iniciales. También será sobescrito el método onUpgrade, que será llamado cuando la base de datos es actualizada, y es donde se pueden añadir, eliminar, modificar tablas... En este ejemplo, se crea una base de datos con la tabla Usuarios, que tendrá dos campos: código (Integer) y nombre (Text).

```
public class UsuariosSQLiteHelper extends SQLiteOpenHelper {

    String sqlCreate = "CREATE TABLE Usuarios (codigo INTEGER, nombre TEXT)";

    public UsuariosSQLiteHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(sqlCreate);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior, int versionNueva) {
        db.execSQL("DROP TABLE IF EXISTS Usuarios");
        db.execSQL(sqlCreate);
    }
}
```

Para manejar la base de datos desde la actividad principal y realizar los procesos de mantenimiento de la misma se ha de instanciar la clase Java, que extiende de SQLiteOpenHelper, e invocamos el método que nos la abre para escritura/lectura. Si es la primera llamada, creará la base de datos con la información pasada al constructor.

```
UsuariosSQLiteHelper usuariosBBDD = new UsuariosSQLiteHelper(this, "DBUsuarios", null, 1);
db = usuariosBBDD.getWritableDatabase();
```



Investiga

El SDK de Android incluye una herramienta de base de datos SQLite, que permite explorar el contenido de las tablas, ejecutar comandos de SQL y llevar a cabo otras funciones de mantenimiento de las bases de datos de SQLite. Investiga y practica.

10.6.1. Insertar registros

Para insertar registros se podrá utilizar el método execSQL, que permite “inyectar” sentencias SQL o usar objetos de la clase ContentValues. En el primer caso, un ejemplo sería:

```
db.execSQL("INSERT INTO Usuarios (codigo,nombre) VALUES (5,'usuarioprueba') ");
```

En el segundo, cada objeto de tipo ContentValues representa una única fila en la tabla y se encarga de emparejar nombres de columnas con valores. Hay que tener en cuenta que, a diferencia de otros motores de bases de datos, los datos en SQLite son débilmente tipados, por lo que es necesario comprobar tipos cuando se asignan o extraen valores de cada columna de una fila.

```
ContentValues nuevoRegistro = new ContentValues();
nuevoRegistro.put("codigo", valor1);
nuevoRegistro.put("nombre", valor2);
db.insert("Usuarios", null, nuevoRegistro);
```

10.6.2. Eliminar registros

Al igual que en el caso anterior, habría dos opciones:

```
db.execSQL("DELETE FROM Usuarios WHERE codigo=5");
-----
db.delete("Usuarios", "codigo=" + valor1, null);
```

10.6.3. Actualizar registros

Utilizando sentencias SQL se procedería de la siguiente forma:

```
db.execSQL("UPDATE Usuarios SET nombre='usunuevo' WHERE codigo=5");
```

O bien mediante ContentValues:

```
ContentValues valores = new ContentValues();
valores.put("nombre", nuevovalor2);
db.update("Usuarios", valores, "codigo=" + valor1, null);
```



En la conferencia Google I/O de 2017 se presentó Room, una librería que abstracta y simplifica el uso de SQLite al implementar una capa intermedia entre la base de datos y el resto de la aplicación. Room funciona con una arquitectura en la que sus clases se marcan con anotaciones preestablecidas y donde la mayoría de las consultas se comprueban en tiempo de compilación.

10.6.4. Consultar registros

Para consultar una base de datos se puede usar el método rawQuery(), que retorna un objeto Cursor que contiene los resultados de la consulta. Este objeto puede tener un número indeterminado de filas, por lo que la forma más fácil de examinar todas sus filas es mediante el método moveToNext() dentro del ciclo while, al que se le pasarán los argumentos buscados. Al recorrer la tabla, se pueden localizar las filas buscadas mediante getString(indice_columna) y getInt(indice_columna); al terminar el recorrido, se ha de llamar al método close() del objeto Cursor.

```
String[] args = new String[] {"5"};
Cursor miCursor = db.rawQuery("SELECT codigo, nombre FROM Usuarios WHERE codigo=? ",args);
txtResultado.setText("");
if (miCursor.moveToFirst()) {
    do { String codigo = miCursor.getString(0);
        String nombre = miCursor.getString(1);
        txtResultado.append(" " + codigo + " - " + nombre + "\n");
    } while(miCursor.moveToNext());
}
miCursor.close();
```

Si se desean mostrar todos los registros, el cursor que se utilizará será el siguiente:

```
Cursor miCursor = db.rawQuery("SELECT codigo, nombre FROM Usuarios",null);
```

Otra alternativa para recuperar registros es usar el método query() de la clase SQLiteDatabase:

A C E G I
 B D F H

Cursor miCursor = db.query(false, tabla, null, null, null, null, null, null, null);

Figura 10.2

Esquema de construcción del método query() de la clase SQLiteDatabase.

Los parámetros que se le han de pasar, de manera obligatoria u opcional, son:

- Booleano que indica si el resultado debe contener valores únicos.
- Nombre de la tabla sobre la que realizar la consulta.

- C. Array de cadenas con el listado de las columnas que se van a incluir en el resultado.
- D. Cláusula WHERE que defina las filas que se van a obtener de la tabla, usando el comodín?
- E. Array de cadenas con los argumentos de selección para la cláusula WHERE.
- F. Cláusula GROUP BY, que indica cómo se agruparán los resultados obtenidos.
- G. Filtro HAVING que limita los grupos que se van a incluir en el resultado.
- H. Cadena que marca la ordenación de las filas obtenidas en el resultado.
- I. Cadena que limita en el número de resultados que se van a devolver.

Un ejemplo de lo anteriormente expuesto, que lista los registros con código=5 de la tabla Usuarios, podría ser el siguiente:

```
String[] args = new String[] {"5"};
String[] campos = new String[] {"codigo", "nombre"};
Cursor miCursor = db.query(false, "Usuarios", campos, "codigo=?", args, null, null, null);
txtResultado.setText("");
if (miCursor.moveToFirst()) {
    do {
        String codigo = miCursor.getString(0);
        String nombre = miCursor.getString(1);
        txtResultado.append(" " + codigo + " - " + nombre + "\n");
    while(miCursor.moveToNext());
    }
    miCursor.close();
}
```



TOMA NOTA

Cuando se han terminado todas las operaciones con la base de datos, no se debe olvidar llamar al método close() del objeto SQLiteDatabase para cerrar la misma.

10.6.5. Movimientos del cursor

Los cursosres tienen una importancia fundamental para tratar la información en una base de datos. Cualquier consulta sobre ella devolverá un objeto de la clase Cursor, que actúa como puntero, al conjunto de resultados buscados. Las posibilidades de desplazamiento que se nos ofrecen son las siguientes:

- moveToFirst: mueve el cursor a la primera fila de los resultados que cumplen la consulta.
- moveToNext: mueve el cursor a la siguiente fila.
- moveToPrevious: mueve el cursor a la fila anterior.
- getCount: devuelve el número de filas que cumplen el resultado de la consulta.
- getColumnNames: devuelve el nombre de la columna que tiene el índice indicado.
- getColumnNames: devuelve un array con el nombre de todas las columnas del cursor.
- moveToPosition: mueve el cursor a la fila indicada.
- getPosition: devuelve el índice de la posición donde se sitúa el cursor.

En los movimientos solicitados (`moveToFirst`, `moveToNext`, `moveToPrevious`, `moveToPosition`), el método devolverá TRUE en caso de haber sido posible realizarlos sin errores.



Actividad propuesta 10.4

Crea una aplicación que tenga una base de datos SQLite, cuya única tabla tenga tres campos, uno de índice y dos de texto. Construye una interface con dos objetos `EditText`, cuatro botones y un `ListView`. Los botones deben tener las siguientes funciones:

- Insertar en la tabla un registro con el contenido de los `EditText`.
- Listar el contenido de la tabla (índice y campos de texto) en el `ListView`.
- Borrar el registro que corresponde a la fila seleccionada en el `ListView`.
- Modificar el registro seleccionado en el `ListView` con la información de los `EditText`.

Antes de borrar y modificar, debe aparecer un `AlertDialog` (con dos botones) que solicite la confirmación de la acción, realizándose esta en caso afirmativo. La realización de las acciones de inserción, modificación y borrado deben ser informadas mediante un `Toast`.

10.7. Almacenamiento en la Red

En este caso, el almacenamiento recae en la parte servidora, mostrándose en el dispositivo móvil aquella información que lee del servidor, por lo que cada acceso requiere una consulta, con alta latencia y sensibilidad a las desconexiones. Uniendo estas técnicas junto con las vistas anteriormente, se puede realizar una persistencia mixta en la que se tiene copia local de parte de la información, realizando esporádicas comunicaciones de sincronización con el servidor.

TEN EN CUENTA

- ✓ Para cualquiera de los métodos que utilicemos de almacenamiento en red es necesario habilitar el `android.permission.INTERNET` en la aplicación.
- ✓ Además, es aconsejable comprobar que tenemos conectividad disponible, para lo cual utilizamos el `android.permission.ACCESS_NETWORK_STATE`, y mediante las clases `ConnectivityManager` y `NetworkInfo` saber si tenemos el servicio conectado (`isConnected()`) y disponible (`isAvailable()`).

10.7.1. Conexión HTTP

Este tipo de conexiones desde aplicaciones Android hacia servidores web siguen el estándar Hipertext Transfer Protocol o HTTP, protocolo con sencillas reglas de transferencia de información entre dispositivos de una red. En la comunicación se realiza una petición de envío,

que suele ir acompañada de datos del cliente (sistema operativo, navegador, archivo solicitado). Una petición puede tener el objetivo de recuperación de datos (GET) o publicación de datos (POST). Android nos ofrece, entre otras, dos posibilidades para interactuar con servidores HTTP/HTTPS:

- *Apache HttpClient*: fácil de usar, especialmente para servicios web REST, pero que a partir de Android 5.1 (Api 22) fue marcada como obsoleta.
- *Clase HttpURLConnection*: basada en la API estándar de Java e incluida en el paquete `java.net.*`, permite desarrollar aplicaciones que contengan clientes HTTP ligeros. Con ella es posible enviar y recibir información a través de la web.



Investiga

Aunque carece de las características de los navegadores a los que estás acostumbrado, Android dispone del objeto `WebView`, que te permite mostrar contenido web como parte del diseño de tu actividad. Resulta muy útil cuando se necesita mayor control sobre la interfaz de usuario, además de permitirte incrustar páginas web en el lugar deseado de tu aplicación. Investiga y practica.

Como primer paso se deberá abrir la conexión hacia el recurso al que queremos acceder en el servidor. Para ello tras instanciar la URL, se usa el método `openConnection()`. El resultado obtenido debe ser casteado a `HttpURLConnection` para instanciar el cliente:

```
URL miUrl = new URL("http://servidor.es/persistencia");
HttpURLConnection miHttp = (HttpURLConnection) miUrl.openConnection();
```

A) Obtener información HTTP

Para recuperar datos de una URL se utiliza el `getInputStream()` para obtener el flujo de datos asociado al recurso buscado, habiendo previamente establecido el tipo de solicitud que faremos mediante el método `setRequestMethod()`. Todo ello dentro de un bloque `try/catch` y dependiendo de la naturaleza de la información descargada puede ser aconsejable hacerlo en un hilo secundario.

```
miHttp.setRequestMethod ("GET");
InputStream miEntrada = miHttp.getInputStream ();
InputStreamReader miLector = new InputStreamReader (miEntrada);
BufferedReader miBuferLector = nuevo BufferedReader (miLector);
Línea = miBuferLector.readLine ();
```

La información recibida de `InputStream` debe ser decodificada para interpretar su contenido. También es importante que, al finalizar la transmisión, se libere la instancia y, con ello, la memoria a ella asociada, mediante el método `miHttp.disconnect()`.

En algunos casos, dependiendo del tipo de servidor, se debe configurar la conexión mediante el método `setRequestProperty`, donde se puede definir el tipo de contenido, cliente, codificación...



SABÍAS QUE...

JSON (Java Script Object Notation) es una buena alternativa como formato de intercambio de datos independiente a la hora de realizar conexiones HTTP.

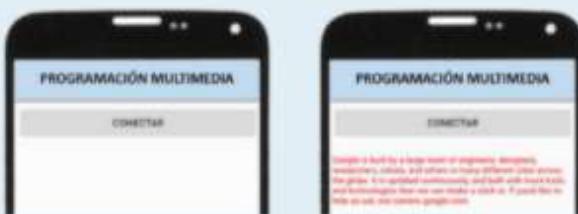
También la clase `HttpURLConnection` nos permite establecer los tiempos de caducidad de conexión, en caso de que esta no se establezca, y tiempos de caducidad al finalizar la lectura. Asimismo, el método `getResponseCode()` nos codifica la respuesta dada por el servidor.

```
miHttp.setConnectTimeout(20000)
miHttp.setReadTimeout(5000)
```



Actividad propuesta 10.5

Realiza un proyecto con una actividad cuya interface tenga un botón y un objeto `TextView`, al pulsar el botón realice una conexión HTTP a la dirección <https://www.google.com/humans.txt> y rellene el `TextView` con el contenido del fichero de texto. No olvides utilizar un hilo secundario y un bloque `try/catch`.



B) Enviar información HTTP

También en un `try/catch`, en este caso tras abrir la conexión, de igual manera que se hizo en el apartado anterior, se establece `setRequestMethod()` como POST y se autoriza la salida de datos con el método `setDoOutput`.

```
miHttp.setRequestMethod ("POST");
miHttp.setDoOutput(true);
```

El siguiente paso consistirá en enviar la información, para lo que se utiliza la clase abstracta `OutputStream`, superclase de todas las que representan flujo de salida. Posteriormente, se abre sobre ella un canal de salida en el que implementar un buffer de escritura.

```
OutputStream miSalida = miHttp.getOutputStream();
OutputStreamWriter miEscritor = new OutputStreamWriter(miSalida);
BufferedWriter miBufferEscritor = new BufferedWriter(miEscritor);
miBufferEscritor.write("Texto a enviar");
```

O bien enviar directamente el flujo de bytes.

```
OutputStream miSalida = new BufferedOutputStream(miHttp.getOutputStream());
String texto = "Texto a enviar";
miSalida.write(texto.getBytes());
miSalida.flush();
miSalida.close();
```



Investiga

Es aconsejable cuidar el formato de codificación de caracteres Unicode e ISO, y aplicárselo correctamente a los métodos lectores y escritores del buffer, lo cual es posible hacer directamente con UTF-8 o mediante la clase StandardCharsets.UTF_8. Investiga y practica.

Si se espera respuesta, se deberá reabrir la entrada para obtener la respuesta del servidor, cerrando la conexión al finalizar con miHttp.disconnect();

WWW

Recurso web

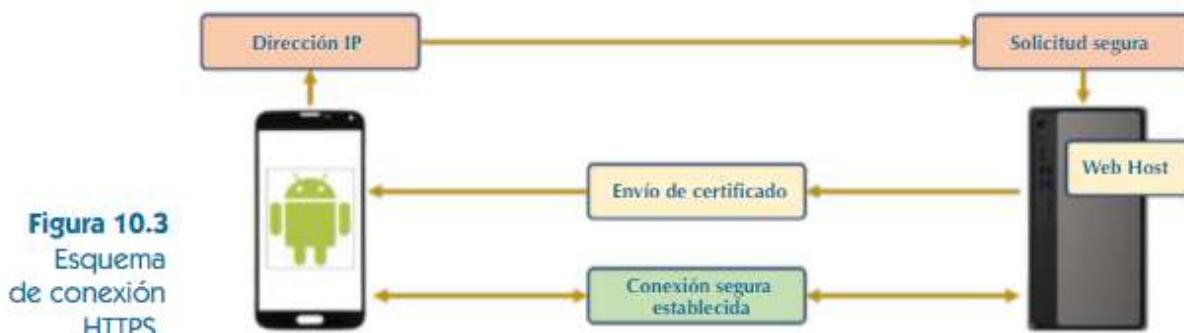
Volley es una librería HTTP desarrollada por Google que facilita, automatiza y optimiza las conexiones HTTP desde las aplicaciones Android hacia servidores externos. Puedes encontrar dicha librería en <https://github.com/google/volley>

10.7.2. Conexión HTTPS

Con el fin de evitar que los datos sean interceptados por terceras partes no deseadas, la capa de conexión segura (SSL y TLS) es fundamental para las comunicaciones encriptadas entre cliente y servidor. Un uso incorrecto de esta capa expondría los datos a entidades maliciosas.

Esto se consigue configurando el servidor con un certificado que contiene una clave pública y una privada coincidente. Como parte del protocolo de enlace entre cliente y servidor, este confirma que tiene clave privada firmando su certificado con criptografía de clave pública.

El cliente debe tener un conjunto de certificados en los que confie, de tal manera que si el certificado del servidor con el que conectar no se encuentra incluido en este conjunto, no se confiará en dicho servidor.



Una manera de simplificar la existencia de certificados es hacer que los servidores se configuren con certificados de emisores conocidos, llamados *autoridades de certificación* (CA). A partir de Jelly Bean, Android posee más de cien autoridades que son actualizadas en cada versión. Por tanto, si nuestra aplicación se conecta a un servidor web con un certificado emitido por una autoridad reconocida, puede realizarse una solicitud segura de la manera anteriormente vista.

En caso contrario, se puede producir una `SSLHandshakeException`, que nos indica que el cliente y el servidor no pudieron negociar el nivel de seguridad deseado. Esto puede ser solucionado indicando a `HttpURLConnection` que confie en un certificado específico, que se obtiene del lugar que se va a visitar con el método `getInstance()`, de la clase `CertificateFactory`. A este método se le pasará como parámetro un estándar para infraestructuras de claves públicas ("X.509").

```

CertificateFactory miFactoria = CertificateFactory.getInstance("X.509");
InputStream entrada = new BufferedInputStream(new FileInputStream(fichero de certificados));
Certificate miAutoridad;
try {
    miAutoridad= miFactoria.generateCertificate(entrada);
} finally {
    entrada.close();
}
    
```

Con esta información se ha de crear un `KeyStore` (almacenamiento para claves y certificados criptográficos) usado para inicializar la interfaz para autoridades de confianza (`TrustManager`), que serán en las únicas en las que confie la aplicación.

```

String tipoClave = KeyStore.getDefaultType();
KeyStore clave = KeyStore.getInstance(tipoClave);
clave.load(null, null);
clave.setCertificateEntry("ca",miAutoridad);
String miAlgoritmo = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory miGestor = TrustManagerFactory.getInstance(miAlgoritmo);
miGestor.init(clave);
    
```

El TrustManager permite inicializar un SSLContext, que sustituye al predeterminado de HttpsURLConnection, usando esta conexión los certificados en él incluidos.

```
SSLContext miContextoSSL = SSLContext.getInstance("TLS");
miContextoSSL.init(null, miGestor.getTrustManagers(), null);

URL miUrl = new URL("https://certs.cac.washington.edu/CAtest/");
HttpsURLConnection miHttp = (HttpsURLConnection)miUrl.openConnection();
miHttp.setSSLSocketFactory(miContextoSSL.getSocketFactory());
```

10.7.3. Conexión por Sockets

En este caso, también se crea un sistema cliente-servidor, donde el servidor abre un Socket que queda a la espera del cliente. Cuando el cliente se conecta, crea un thread para esa comunicación y, por medio de un bucle, se mantiene hasta que un evento determinado la finaliza.

En la aplicación utilizada como cliente se usará la clase `Socket`, mientras que del lado servidor se trabajará con uno o más objetos de la clase `Socket` asociados a un `ServerSocket`, realizando las entradas y salidas a través de objetos `InputStream` y `OutputStream`.

A) Servidor Sockets

Al igual que en otros procesos que pueden frenar el funcionamiento del hilo principal, y más teniendo en cuenta la funcionalidad de la aplicación, esta debe ser desarrollada en un hilo secundario, siguiendo los pasos trabajados en el capítulo anterior. Asimismo, los procesos de lectura y escritura deben ir incluidos en un bloque `try/catch` para controlar excepciones.

Como primer paso para la construcción de un servidor `Socket`, se deberá crear un `ServerSocket` con el número de puerto como parámetro, que quedará a la espera de las conexiones de los clientes mediante el método `accept()` de la clase `Socket`, que estará bloqueado hasta que un cliente se conecte. Un servidor puede realizar múltiples conexiones con clientes.

RECUERDA

- ✓ Los puertos utilizados para el Socket Servidor pueden estar comprendidos entre el 1024 y el 65535 (del 1 al 1023 están reservados para servicios del sistema). Cada servidor debe usar un puerto diferente.

```
ServerSocket miServerSocket;
miServerSocket = new ServerSocket(8080);
Socket miSocket = miServerSocket.accept();
```

Realizada la petición, este método devuelve un objeto `Socket`, que representa la conexión recién abierta, y nos puede informar de la dirección IP del cliente y de su puerto de comunicación.

```
ipCliente= miSocket.getInetAddress()
puertoCliente= miSocket.getPort()
```

Con el cliente conectado, se obtiene o envía información a través de él mediante los métodos `getOutputStream()` o `getInputStream()`. Estos métodos tan solo son útiles para enviar bytes, por lo que para enviar otro tipo de información podemos recurrir a varias alternativas.

Una es instanciar un buffer de lectura o escritura sobre `InputStreamReader` o `OutputStreamWriter`, ambos llevando como parámetro los métodos anteriormente comentados.

```
BufferedWriter miEscritor = new BufferedWriter(new OutputStreamWriter(miSocket.getOutputStream()));
miEscritor.write ("texto");
miEscritor.flush(); //Fuerza la escritura
miEscritor.close();
BufferedReader miLector= new BufferedReader(new InputStreamReader(miSocket.getInputStream()));
entrada = miLector.readLine();
miLector.close();
```

Otra posibilidad es mediante las clases `DataInputStream` y `DataOutputStream`. Su constructor admite `InputStream` y `OutputStream` y tiene sus propios métodos de lectura o escritura (`readline()`, `readInt()`, `writeInt()`, `writeChar()`, `writeFloat()`...).

```
DataInputStream miLector = new DataInputStream(miSocket.getInputStream());
entrada = miLector.readLine();
miLector.close();
```

Investiga

La clase `PrintWriter` puede ser usada para dar salida por el Socket, utilizando el método `getOutputStream()` como parámetro de la clase. Investiga y practica.

```
PrintWriter salida = new PrintWriter(miSocket.getOutputStream());
salida.println("Cadena para enviar");
salida.flush();
```

B) Cliente Sockets

Se instancia el Socket utilizando la dirección IP del servidor y el puerto que este emplea para la escucha. Posteriormente, le sigue alguna de las posibilidades de lectura o escritura vistas en el apartado del servidor.

```
Socket miSocket = new Socket("192.168.1.134", 8080);
```

Es importante recordar que, al recogerse la información en un hilo secundario, deben utilizarse algunos de los métodos anteriormente vistos (`handler`, `runOnUiThread`) para llevar esta información al hilo principal.

Actividad propuesta 10.6

Realiza dos proyectos con una actividad cada uno. El primero actuará como Servidor Socket, tendrá dos botones y tres campos de texto: el primer botón activará el servidor y el segundo lo detendrá, el servidor recibirá del cliente tres líneas de información que mostrará en los respectivos campos de texto. Cuando reciba la información, enviará un mensaje al cliente diciendo que la información ha sido recibida.

El cliente tendrá tres objetos EditText, un botón y un campo de texto. Al presionar el botón, enviará al servidor el contenido de los EditText; si la emisión ha sido correcta, mostrará la respuesta recibida del servidor en el campo de texto.

**RECUERDA**

- ✓ Como seguramente hayas estudiado en el módulo de Acceso a Datos, es posible realizar persistencia en la nube mediante SDK o servicios API Rest. Para lo cual podemos hacer uso de Firebase, MongoDB, ParseData, CouchDB...

10.8. Proveedores de contenidos

Un proveedor de contenido es un componente de Android que permite a una aplicación gestionar el acceso a los datos almacenados por ella misma u otra aplicación. Controlan el acceso, encapsulan la información y suministran mecanismos para definir la seguridad de la misma.

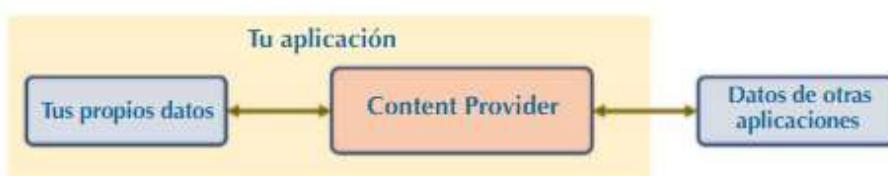


Figura 10.4
Esquema de funcionamiento de un Content Provider.

Muchas de las aplicaciones Android (agenda, contactos, calendario) utilizan este mecanismo como medio de exposición de los datos por ellas gestionados, lo que nos permite acceder a esta información desde nuestras propias aplicaciones. Esto hace suponer que son dos apartados los que habrá que desarrollar para su conocimiento, por un lado, el mecanismo de acceso a fuentes de datos ya existentes y, por otro, el crear un proveedor personalizado de datos.

10.8.1. Crear un proveedor de datos (Content Provider)

El proveedor de contenidos actuará como una base de datos relacional, donde podemos realizar operaciones de insertar, actualizar, eliminar y editar los datos almacenados en la misma.

Para ello, será necesario disponer de un método de almacenamiento que contenga la información que se desea compartir (ficheros internos, XML, SQLite), almacenamiento que debe contar con un campo índice que identifique cada uno de los registros. Como ejemplo para este apartado, puede ser utilizada la siguiente clase, que crea y rellena una base de datos SQLite.

```
public class TablaSqliteHelper extends SQLiteOpenHelper {
    String sqlCreate = "CREATE TABLE Tabla " + "(" + "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        " campo1 TEXT, " + " campo2 TEXT, " + " campo3 TEXT, " + " campo4 TEXT )";

    public TablaSqliteHelper (Context contexto, String nombre, CursorFactory factory, int version) {
        super(contexto, nombre, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(sqlCreate);
        //se rellena la tabla con 10 registros
        for(int i=1; i<=10; i++) {
            {
                String campo1 = "Dato1-" + i;
                String campo2 = "Dato2-" + i;
                String campo3 = "Dato3-" + i;
                String campo4 = "Dato4-" + i;

                db.execSQL("INSERT INTO Tabla (campo1, campo2, campo3, campo4) " +
                    "VALUES ('" + campo1 + "', '" + campo2 + "', '" + campo3 + "', '" + campo4 + "')");
            }
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior, int versionNueva) {
        db.execSQL("DROP TABLE IF EXISTS Tabla");
        db.execSQL(sqlCreate);
    }
}
```

Ya con la fuente de datos operativa, se ha de crear una clase que extienda de ContentProvider, registrarla en AndroidManifest.xml (etiqueta <provider>) y definir el URI (uniform resource identifier) del proveedor de contenidos para acceder al mismo. Este URI, estará formado por el prefijo “content://”, el identificador del Content Provider (también llamado *authority*) y, por último, la entidad a la que se quiera acceder en los datos proporcionados por el Content Provider.

La clase creada deberá tener los siguientes métodos: onCreate(), query(), insert(), update(), delete() y getType(). En el primero se inicializan los recursos y el último identifica el tipo de datos (MIME) devueltos, los demás se centran en labores de mantenimiento de los datos.

Junto a los métodos, será necesario definir una serie de constantes necesarias para la creación del proveedor. Una de ellas es la autoridad y otra el contenido.

```
private static final String AUTHORITY ="es.ejemplo.android.proyecto6 ";
public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + "tabla");
```

La clase UriMatcher permite interpretar patrones en la URI, y nos permite distinguir cuándo una URI hace referencia a varias filas de la tabla o a un registro concreto.

- “content://es.ejemplo.android.proyecto5/tabla” - Acceso a toda la tabla.
- “content://es.ejemplo.android.proyecto5/tabla/10” - Acceso al registro 10 de la tabla.

Esto se hará definiendo dos nuevas constantes, una para cada caso, que serán añadidas a una instancia de la clase UriMatcher.

```
private static final int TABLA TODOS = 1;
private static final int TABLA UNO = 2;
private static final UriMatcher miUriMatcher;

static {
    miUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    miUriMatcher.addURI("es.ejemplo.android.proyecto6", "tabla", TABLA TODOS);
    miUriMatcher.addURI("es.ejemplo.android.proyecto6", "tabla#", TABLA UNO);
}
```

También se definirán las constantes con los nombres de los campos (columnas) de los datos que suministra el Content Provider y que debe tener la fuente de datos.

```
public static final class Tabla implements BaseColumns
{
    private Tabla() {}
    public static final String COL_CAMP01 = "campo1";
    public static final String COL_CAMP02 = "campo2";
    public static final String COL_CAMP03 = "campo3";
    public static final String COL_CAMP04 = "campo4";
}
```

Para terminar, se definirán los atributos necesarios para almacenar el nombre de la base de datos, la versión y la tabla a la que accederá el Content Provider.

```
private TablaSqliteHelper tablabbdd;
private static final String BD_NOMBRE = "DBaseDatos";
private static final int BD_VERSION = 1;
private static final String TABLA = "Tabla";
```

Con lo anterior ya tenemos configurado el proveedor de contenidos, el siguiente paso será implementar y dotar de contenido los diferentes métodos del mismo. En onCreate se inicializará la base de datos, con su nombre y versión. Para ello utilizaremos la clase creada como fuente de datos (TablaSqliteHelper).

```
@Override
public boolean onCreate() {
    tablabbdd = new TablaSqliteHelper(
        getContext(), BD_NOMBRE, null, BD_VERSION);
    return true;
}
```

El método query() es el verdadero motor del proveedor, recibiendo como parámetros la URI, una array de nombres de columnas, los criterios y argumentos de selección y la forma de ordenación. Deberá devolver la información solicitada según lo indicado con estos parámetros.

Dentro de este medio será necesario distinguir si se solicita información de todos los registros o de una fila en concreto, lo que se consigue mediante el método match() del objeto UriMatcher. También necesitaremos el método getLastPathSegment() para extraer el id de la tabla. Realizada la consulta a la base de datos, se recibirán los datos en forma de cursor.



SABÍAS QUE...

Android implementa proveedores de contenido que suministran datos, como direcciones, contactos personales, imágenes, vídeo, audio..., a los que cualquier aplicación puede acceder bajo ciertas limitaciones. Puedes obtener información de ellos en la página de Android Developer ([android.provider](#)).

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    String where = selection;
    if(miUriMatcher.match(uri) == TABLA_UNO){
        where = "_id=" + uri.getLastPathSegment();
    }
    SQLiteDatabase db = tablabbdd.getWritableDatabase();
    Cursor miCursor = db.query(TABLA, projection, where, selectionArgs, null, null, sortOrder);
    return miCursor;
}
```

Los métodos de update() y delete() son similares en su construcción, con la diferencia de que nos devuelven el número de registros en lugar de un cursor de los resultados:

```
@Override
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    int cont;
    String where = selection;
    if(miUriMatcher.match(uri) == TABLA_UNO){
        where = "_id=" + uri.getLastPathSegment();
    }
    SQLiteDatabase db = tablabbdd.getWritableDatabase();
    cont = db.update(TABLA, values, where, selectionArgs);
    return cont;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    int cont;
    String where = selection;
    if (miUriMatcher.match(uri) == TABLA_UNO) {
        where = "_id=" + uri.getLastPathSegment();
    }
    SQLiteDatabase db = tablabbdd.getWritableDatabase();
    cont = db.delete(TABLA, where, selectionArgs);
    return cont;
}
```

El método insert() devuelve la URI que apunta al registro insertado. El id se obtendrá de SQLiteDatabase y la nueva URI se construirá mediante el método ContentUris.withAppendedId(), que debe recibir como parámetro la URI del Content Provider y el id del elemento añadido.

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    long idnueva = 1;
    SQLiteDatabase db = tablabbdd.getWritableDatabase();
    idnueva = db.insert(TABLA, null, values);
    Uri nuevaUri = ContentUris.withAppendedId(CONTENT_URI, idnueva);
    return nuevaUri;
}
```

Por último, como se ha comentado, el método getType() identifica el tipo de datos que devuelve el Content Provider, expresados como un MIME Type, lo que será útil a Android para conocer qué aplicaciones pueden procesar dichos datos. Son posibles dos tipos MIME, uno de ellos en caso de ser devuelta una lista de registros y otro en caso de devolverse un solo registro.

```
@Override
public String getType(Uri uri) {
    int match = miUriMatcher.match(uri);
    switch (match)
    {
        case TABLA_TODO:
            return "vnd.android.cursor.dir/ejemplo.android.proyecto6";
        case TABLA_UNO:
            return "vnd.android.cursor.item/ejemplo.android.proyecto6";
        default:
            return null;
    }
}
```

La inclusión en el AndroidManifest debe ser de la siguiente forma:

```
<provider
    android:name="Actividad2"
    android:authorities="es.ejemplo.android.proyecto6"/>
```

Actividad propuesta 10.7



En la plataforma de Editorial Síntesis encontrarás el código completo de la aplicación basada en Content Provider elaborada en este apartado. Práctica con él.

10.8.2. Acceder a datos de otra aplicación (Content Resolver)

Este componente es el responsable de acceder a la información del Content Providers. El objeto Content Resolver se comunicará con el objeto proveedor que será implementado por instancia de clase, para lo que debe analizar la URI suministrada y realizar una búsqueda hasta

llegar a la base de datos especificada. El objeto CursorLoader será utilizado para ejecutar la consulta de forma asíncrona en segundo plano. La aplicación principal llamará a un CursorLoader para obtener los datos deseados del Content Provider usando Content Resolver.



Figura 10.5
Esquema de funcionamiento de un Content Resolver.

Para acceder a un proveedor, debemos obtener una referencia a un Content Resolver (utilizando el método `getContentResolver()`), mediante el cual interactuaremos con el Content Provider. Una vez se tenga esta referencia, podrán ser utilizados los métodos `query()`, `update()`, `insert()` y `delete()` para realizar dichas acciones equivalentes sobre el Content Provider.

A) Realizar consultas

Muy similar a las consultas a bases de datos SQLite. Habrá que definir un array con los nombres de las columnas que se van a recuperar al hacer la consulta. Los resultados se obtendrán en forma de cursor.

El método `query()` recibirá la URI del Content Provider, el array de columnas, el criterio y argumentos de selección, y el criterio de ordenación de los resultados.

```

String[] columnas = new String[] {
    Actividad2.Tabla._ID,
    Actividad2.Tabla.COL_CAMPO1,
    Actividad2.Tabla.COL_CAMPO2,
    Actividad2.Tabla.COL_CAMPO3,
    Actividad2.Tabla.COL_CAMPO4};
Uri tablaUri = Actividad2.CONTENT_URI;
ContentResolver miResolver = getContentResolver();
Cursor miCursor = miResolver.query(tablaUri, columnas, null, null, null);
  
```

Una vez obtenido el cursor, lo recorreremos para extraer los resultados. Si solo quisiéramos listar el contenido del índice y el Campo 2, el código sería el siguiente:

```

String[] columnas = new String[]{Actividad2.Tabla._ID,Actividad2.Tabla.COL_CAMPO2};
Uri tablaUri = Actividad2.CONTENT_URI;
ContentResolver miResolver = getContentResolver();
Cursor miCursor = miResolver.query(tablaUri, columnas, null, null, null);
if (miCursor.moveToFirst()) {
    String indice, campo2;
    int colIndice = miCursor.getColumnIndex(Actividad2.Tabla._ID);
    int colCampo2 = miCursor.getColumnIndex(Actividad2.Tabla.COL_CAMPO2);
    texto.setText("");
    do {
        indice = miCursor.getString(colIndice);
        campo2 = miCursor.getString(colCampo2);
        texto.append(indice + " - " + campo2 + "\n");
    } while (miCursor.moveToNext());
}
  
```

Otra alternativa para la obtención de la URI podría ser:

```
String URL = "content://es.ejemplo.android.proyecto6";
Uri tablaUri = Uri.parse(URL);
```

B) Insertar registros

Al igual que se hace con SQLite, se rellena, en primer lugar, un objeto ContentValues con los datos que se van a insertar, para luego usar el método insert(), al que se le ha de pasar la URI del Content Provider y los datos del nuevo registro.

```
ContentValues misContent = new ContentValues();
misContent.put(Actividad2.Tabla.COL_CAMPO1, "Nuevo Valor 1");
misContent.put(Actividad2.Tabla.COL_CAMPO2, "Nuevo Valor 2");
misContent.put(Actividad2.Tabla.COL_CAMPO3, "Nuevo Valor 3");
misContent.put(Actividad2.Tabla.COL_CAMPO4, "Nuevo Valor 4");
ContentResolver miResolver = getContentResolver();
String URL = "content://es.ejemplo.android.proyecto6";
Uri tablaUri = Uri.parse(URL);
miResolver.insert(tablaUri, misContent);
```

C) Borrar registros

Para eliminar un registro se utilizará directamente el método delete() del Content Resolver, al que habrá que indicar el criterio de localización del registro que se va a eliminar.

```
ContentResolver miResolver = getContentResolver();
String URL = "content://es.ejemplo.android.proyecto6";
Uri tablaUri = Uri.parse(URL);
miResolver.delete(tablaUri, Actividad2.Tabla.COL_CAMPO2 + " = 'Nuevo Valor 2'", null);
```

Actividad propuesta 10.8



En la plataforma de Editorial Síntesis encontrarás el código completo de la aplicación basada en Content Resolver elaborada en este apartado. Práctica con él.

D) Acceso a Content Provider públicos

Android dispone de información, en el dispositivo, a la que es posible acceder mediante esta técnica. Entre la que se expone en la documentación del paquete android.provider, se encuentra la posibilidad de acceder al registro de llamadas situado en android.provider.CallLog. De él se pueden extraer variados datos, de los que se van a utilizar, en el siguiente ejemplo, el

tipo de llamada (CallLog.Calls.TYPE) y el número de dicha llamada (CallLog.Calls.NUMBER).

El proceso es idéntico al visto en los puntos anteriores, obteniendo el URI, instanciando un ContentResolver y recuperando de él un cursor con unos criterios de búsqueda.

Mediante el tipo es posible conocer si se trata de una llamada recibida (Calls.INCOMING_TYPE), emitida (Calls.OUTGOING_TYPE) o perdida (Calls.MISSED_TYPE).

```
String[] columnas = new String[]{CallLog.Calls.TYPE, CallLog.Calls.NUMBER};
Uri llamadasUri = CallLog.Calls.CONTENT_URI;
ContentResolver miResolver = getContentResolver();
Cursor miCursor = miResolver.query(llamadasUri, columnas, null, null, null);

if (miCursor.moveToFirst()) {
    int tipo;
    String tipoLlamada = "";
    String numeroTele;
    int colTipo = miCursor.getColumnIndex(CallLog.Calls.TYPE);
    int colNumero = miCursor.getColumnIndex(CallLog.Calls.NUMBER);
    texto.setText("");
    do {
        tipo = miCursor.getInt(colTipo);
        numeroTele = miCursor.getString(colNumero);
        if (tipo == CallLog.Calls.INCOMING_TYPE) tipoLlamada = "RECIBIDA";
        else if (tipo == CallLog.Calls.OUTGOING_TYPE) tipoLlamada = "EMITIDA";
        else if (tipo == CallLog.Calls.MISSED_TYPE) tipoLlamada = "PERDIDA";
        texto.append(tipoLlamada + " - " + numeroTele + "\n");
    } while (miCursor.moveToNext());
}
```

Para que esto sea posible, es necesario autorizar la consulta en el AndroidManifest.xml, y en caso de utilizar Marshmallow o versiones superiores, gestionar los permisos en el propio dispositivo.

```
<uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
<uses-permission android:name="android.permission.READ_CALL_LOG"></uses-permission>
```



Actividad propuesta 10.9

Crea una base de datos SQLite con tres campos, uno índice, otro que almacene el nombre de la versión de Android y un tercero para el año de lanzamiento. Elabora un Content Provider que posibilite el acceso y gestión de esta información.

Crea una actividad con dos campos EditText, tres botones y un GridView. Un botón permite mostrar en el GridView la información suministrada por el Content Provider. Otro botón elimina de la base de datos el registro correspondiente a la celda seleccionada en el GridView. Y el tercer botón añadiría un registro con la información suministrada en EditText.

Resumen

- Se denomina *persistencia* a la capacidad que debe tener una aplicación para que los datos por ella manipulados perduren en el tiempo tras la ejecución de la misma.
- La configuración o determinados aspectos de una aplicación pueden conservarse y compartirse mediante la clase SharedPreferences en formato de pares de clave-valor.
- En Android es posible utilizar como medio donde hacer persistencia, la memoria interna, con las limitaciones que esta nos impone, y la tarjeta de almacenamiento, con posibles riesgos de seguridad. Además, los recursos son una buena fuente de información, si bien no es posible utilizarlos como fuente de almacenamiento dinámico.
- Una buena alternativa a las anteriores, especialmente si aumenta el volumen de la información, es el uso de bases de datos. Entre los gestores disponibles, SQLite, por su tamaño, características e integración en Android es uno de los más usados. Este gestor nos permite realizar todo tipo de labores de mantenimiento de la información (añadir, borrar, mostrar, modificar...).
- La persistencia mixta es aquella que combina algunas de las técnicas locales anteriormente vistas, junto con el almacenamiento en red, donde una parte importante de las labores de almacenamiento recaen sobre la parte servidora. Para ello, podemos utilizar diferentes métodos de conexión, y casi todos ellos con variantes (HTTP/HTTPS, Socket), permitiéndonos Android desarrollar aplicaciones que desempeñen funciones tanto de cliente como de servidor.
- Un proveedor de contenido es un componente de Android que permite a una aplicación gestionar el acceso a los datos almacenados por ella misma u otra aplicación. Gestiona el acceso, encapsula la información y suministra mecanismos para definir la seguridad de la misma.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Con respecto a la clase SharedPreferences:
 - a) Permite guardar y recuperar en formato de pares de clave-valor.
 - b) Se utiliza el método getSharedPreferences() cuando se necesita un archivo de preferencias.
 - c) Se utiliza el método getPreferences () cuando se necesitan muchos archivos de preferencias.
 - d) Todas las opciones son correctas.
2. ¿Qué modo de acceso usamos para el almacenamiento en la memoria interna del dispositivo?:
 - a) MODE_PRIVATE.
 - b) MODE_PUBLIC.
 - c) MODE_READABLE.
 - d) MODE_WRITEABLE.

3. ¿Qué método nos puede informar de la situación de la tarjeta externa de almacenamiento?:

- a) Environment.getExternalStorage().
- b) Environment.getExternalStorageState().
- c) Environment.getSDStorageState().
- d) Environment.getExternalStorageState().

4. ¿Qué clase gestiona la base de datos SQLite?:

- a) CursorFactory.
- b) SQLiteOpenHelper.
- c) SQLiteCursor.
- d) SQLiteDatabase.

5. ¿Qué método de HttpURLConnection permite informar al servidor del cliente utilizado?:

- a) setRequestMethod().
- b) getResponseCode().
- c) setRequestProperty().
- d) setDoOutput().

6. ¿A partir de qué versión de Android se incorpora una lista de autoridades certificadoras?:

- a) KitKat.
- b) Lollipop.
- c) Jelly Bean.
- d) Marshmallow.

7. ¿Qué método deja a un servidor Socket en espera de la conexión de un cliente?:

- a) miServerSocket.wait().
- b) miServerSocket.standby().
- c) miServerSocket.accept().
- d) miServerSocket.client().

8. ¿Qué parámetros se necesitan al instanciar un cliente Socket?:

- a) IP Server, Port Server.
- b) IP Server, IP Client.
- c) IP Server, Port Client.
- d) IP Cliente, Port Client.

9. ¿Qué utilidad tiene la clase UriMatcher?:

- a) Define la URI de un Content Provider.
- b) Interpreta patrones en la URI.
- c) Define el tipo de datos (MIME) devueltos por la URI.
- d) Identifica a la autoridad proveedora de contenidos.

10. ¿Cuál de los siguientes tipos de llamadas no puede ser devuelto por CallLog.Calls.TYPE?:

- a) Calls.INCOMING_TYPE.
- b) Calls.OUTGOING_TYPE.
- c) Calls.RINGING_TYPE.
- d) Calls.MISSED_TYPE.

SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d