DESARROLLO DE APLICACIONES MÓVILES CON ANDROID

Manuel Torres Remon

EDITORIAL MACRO®

Perú - México - Colombia - Chile - Ecuador - España - Bolivia - Uruguay - Guatemala - Costa Rica

EDITORIAL MACRO®

Desarrollo de aplicaciones móviles con Android
Autor: Manuel Torres Remon

© Derechos de autor registrados:
Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:
Empresa Editora Macro EIRL

Coordinadora de edición:
Magaly Ramon Quiroz

Diseño de portada:
Fernando Cavassa Repetto

Diseño y diagramación:
Katia Valverde

Edición a cargo de:
© Empresa Editora Macro EIRL
Av. Paseo de la República N.° 5613, Miraflores, Lima, Perú

Teléfono: (511) 748 0560
E-mail: proyectoeditorial@editorialmacro.com
Página web: www.editorialmacro.com

Primera edición: Julio 2019
Tiraje: 1000 ejemplares

Impresión:
Talleres gráficos de la Empresa Editora Macro EIRL
Jr. San Agustín N.° 612-624, Surquillo, Lima, Perú
Julio 2019

ISBN N.° 978-612-304-574-6
Hecho el Depósito Legal en la Biblioteca Nacional del Perú N.° 2019-08440

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.

MANUEL TORRES REMON

Licenciado en Informática. Se ha dedicado a la consultoría y docencia de cursos de tecnología desde hace 17 años. A su vez, ha brindado diversos talleres de capacitación en las instituciones más importantes de Lima (Perú).

Estudió las carreras de Ingeniería de Sistemas (Universidad Peruana del Norte), Educación (Universidad Alas Peruanas) y Computación e Informática (Instituto de Educación Superior Manuel Arévalo Cáceres). En todas estas instituciones recibió una esmerada formación profesional que ha podido demostrar ampliamente a lo largo de su trayectoria docente.

Actualmente, se desempeña como docente de tecnología en instituciones de educación superior, como Senati, Cibertec, Manuel Arévalo Cáceres y Unimaster, donde imparte los cursos de Programación, Base de Datos y Análisis de Sistemas.

Dedicatoria

Este libro está dedicado con mucho cariño y aprecio a mis pequeñas Ángela Victoria y Fernanda Ximena, que son y seguirán siendo mi fuente de inspiración, y mucho más a mi esposa Luz, por comprenderme en todo lo que me propongo.

Agradecimientos

La producción de todo libro es un sacrificio que debe ser valorado y eso solo ocurre si usted como lector aprecia este material tal como lo aprecio yo mismo. Escuché alguna vez la siguiente frase: “La lectura de un libro enriquece de conocimientos y empobrece la ignorancia”. No obstante, en la actualidad, un libro impreso es difícil de leer, pues son pocas las personas que cargan libros en la mano, ya que estos han sido reemplazados por los dispositivos móviles. En consecuencia, mi primer agradecimiento es para usted, amigo lector, por adquirir este material que me ha demandado sacrificio, tiempo y esfuerzo. Finalmente, deseo agradecer a la gran familia Macro por confiar nuevamente en mí para el desarrollo del presente libro.

Índice

Introducción 15

■ Capítulo 1: Plataforma Android 17

1.1 Introducción al Android 19
1.2 Historia de Android 20
1.3 Open Handset Alliance TM 21
1.4 Estructura Android 24
1.5 Características de Android 26
1.6 Arquitectura Android (Librerías) 26
1.6.1 Zygote 27
1.6.2 Davilk Virtual Machine 27
1.6.3 ART – Android RunTime 27
1.6.4 File System 27
1.7 Android SDK 27
1.8 Tipos de procesos de Android 28
1.9 Versiones Android 29
1.10 Entorno de desarrollo 30
1.10.1 Android Studio 30
1.11 Fundamentos de aplicaciones Android 36
1.11.1 Estructura de un proyecto en Android Studio 36
1.11.2 Definición de Activity 37
1.11.3 Ciclo de vida de un activity 38
1.11.4 Agregar un activity 39
1.11.5 Eliminar un activity 39
1.11.6 Modificar la actividad de inicio del proyecto 40
1.11.7 Modificar el tamaño y tipo de letra del editor en Android Studio 41
1.11.8 Uso de los recursos 41
1.12 Estudio del archivo Android Manifest 42
CASO DESARROLLADO 01: Interfaz inicial con Android Manifest 43

■ Capítulo 2: Fundamentos Java para Android 49

2.1 Tipos de datos 51
2.2 Variables 52
2.3 Programación orientada a objetos 53
2.4 Clase-objeto 54

2.4.1 Crear una clase dentro del proyecto Android 55
2.4.2 Modificar el nombre de la clase 56
2.4.3 Eliminar una clase 56
2.5 Declarar objetos de controles 57
2.6 Inicialización de variables 57
2.7 Conversiones 58
2.8 Casting 59
2.9 Atributos 60
2.10 Métodos get y set 60
2.10.1 Método set 60
2.10.2 Método get 61
2.10.3 Implementación de métodos get y set con Android Studio 62
2.11 Métodos definidos por el usuario 65
2.11.1 Métodos sin valor de retorno 65
2.11.2 Método con valor de retorno 66
2.10.3 Método constructor 66
CASO DESARROLLADO 01: Manejo de variables 67
CASO DESARROLLADO 02: Programación orientada a objetos 71

■ Capítulo 3: Recursos y componentes Android 77

3.1 Estructura del proyecto Android 79
3.2 Recursos 80
3.2.1 Mipmap 80
3.2.2 Drawables 83
3.2.3 Layouts 85
3.2.4 Values 87
3.3 Ciclo de vida de un activity 89
3.3.1 Ciclo de vida 90
3.3.2 Estados 91
3.3.3 Navegación 93
3.3.4 Uso del método onCreate de la actividad 95
3.4 Intents en la gestión de actividades 95
3.5 Manejo de Intent Filter 97
3.6 Uso de PendingIntent 98
3.7 Orientación de la pantalla 99
3.8 Tamaños de la pantalla 100
3.9 Creación de fragmentos 102
3.10 Registrar fragmentos en la aplicación 104
CASO DESARROLLADO 01: Asignar un ícono a nuestra aplicación 105
CASO DESARROLLADO 02: Asignar imagen de fondo a la aplicación 109
CASO DESARROLLADO 03: Asignar colores de fondo 110
CASO DESARROLLADO 04: Asignar títulos a la aplicación 113

CASO DESARROLLADO 05: Manejo de arreglos de textos 114
CASO DESARROLLADO 06: Asignación de estilos 118
CASO DESARROLLADO 07: Manejo de actividades 122

■ Capítulo 4: Controles básicos y notificaciones para aplicaciones Android 131

4.1 Interfaces de usuario 133
4.2 Densidad y tamaño de la pantalla 136
4.3 Clase View 136
4.4 Distribución de los ViewGroups 137
4.5 Control LinearLayout 138
4.6 Control TableLayout 140
4.7 Control FrameLayout 141
4.8 Control TextView 142
4.9 Control EditText 143
4.10 Control Button 144
4.11 Control ScrollView 147
4.12 Control ImageView 149
4.13 Control ImageButton 150
4.14 Control Spinner 151
4.15 Control CheckBox 152
4.16 Control ProgressBar 153
4.17 Dialog Window 154
4.18 Progress Dialog 155
4.19 Diálogos personalizados 156
4.20 Notificaciones con Toast 164
4.21 Menú de opciones 166
CASO DESARROLLADO 01: Manejo de LinearLayout Horizontal 170
CASO DESARROLLADO 02: Manejo de LinearLayout Vertical 175
CASO DESARROLLADO 03: Manejo de TableLayout 177
CASO DESARROLLADO 04: Manejo de FrameLayout 180
CASO DESARROLLADO 05: Manejo de la clase CheckBox 181

■ Capítulo 5: Controles avanzados para aplicaciones Android 185

5.1 Orientación del dispositivo 187
5.2 Control ListView 187
5.3 Control CardView 190
5.4 Control TabLayout 192
5.5 Control ToolBar 198
CASO DESARROLLADO 01: Manejo de ListView 201
CASO DESARROLLADO 02: Manejo de CardView 209
CASO DESARROLLADO 03: Manejo de TabLayout 222

■ Capítulo 6: Manejo de archivos 235
6.1 Persistencia de datos: Files 237
6.1.1 Preferencias compartidas 237
6.1.2 Almacenamiento interno 237
6.1.3 Almacenar archivos en memoria externa 238
6.2 Clase File 238
6.3 Clase FileOutputStream 239
6.4 Clase FileInputStream 240
6.5 Leer XML Data 241
CASO DESARROLLADO 01: Manejo de la clase FileOutputStream 242
CASO DESARROLLADO 02: Manejo de la clase InputStream 244
CASO DESARROLLADO 03: Leer XML 247

■ Capítulo 7: Acceso a datos con SQLite 253
7.1 Fundamentos SQLite 255
7.2 Gestión de base de datos SQLite 256
7.3 Componente SQLiteOpenHelper 256
7.4 Métodos de inserción, actualización y eliminación 257
7.5 Clase ContentValues 257
7.6 Cursores y contenedores de valores en SQLite 258
7.7 Modelo Vista Controlador 258
CASO DESARROLLADO 01: Registro y listado de productos 259
CASO DESARROLLADO 02: CRUD para clientes 265

■ Capítulo 8: Networking y servicios en Android 281
8.1 Qué es un servicio 283
8.2 Intent Service 284
8.3 Bound Service 285
8.4 Configurar el archivo de manifiesto 285
CASO DESARROLLADO 01: Servicios de mensajes 286
CASO DESARROLLADO 02: Generación de números aleatorios con servicios usando Bound Service .. 291

■ Capítulo 9: Servicios basados en localización 295
9.1 Google Maps 297
9.2 Manejo de las API de Google Maps 297
CASO DESARROLLADO 01: Ubicación Actual en el Google Maps 299

■ Capítulo 10: Multimedia en Android 303
10.1 Componente MediaPlayer 305
10.2 Visualización de videos 306
CASO DESARROLLADO 01: Reproductor de audio (MP3) 307
CASO DESARROLLADO 02: Visor de videos desde Youtube 310
CASO DESARROLLADO 03: Visor de video desde Youtube usando credenciales 314
CASO DESARROLLADO 04: Listado de audios 319

■ Capítulo 11: Publicación en Google Play 323
11.1 Generación de APK con certificado digital 325
11.2 Creación de una cuenta en Google Play 326
11.3 Publicación de la aplicación 327

Introducción

La revolución digital empezó con la llegada de Internet hace algunos años y más aun con la masificación de los teléfonos móviles, pero cuando se juntan ambas tecnologías, se aprecia un futuro distinto y tecnológico, como si fuera una película futurista.

Durante muchos años, se comentó sobre la nueva era de la digitalización o revolución digital, y esta llegó aproximadamente en el año 2015 con la invasión de los dispositivos móviles. En esos años, se trataba sobre las coberturas con 3G, 4G, pero, en la actualidad, ya nos encontramos en la 5G. La velocidad de transmisión mejoró considerablemente, y los dispositivos móviles se presentan en diversos aparatos como tablets, televisores, etc. Nosotros no estamos ajenos a ello y nos preparamos para recibir esta nueva tecnología. En este sentido, esta publicación se presenta como una opción de inicio en la programación para dispositivos móviles. Por ese motivo, hemos preparado el material en once capítulos, totalmente prácticos y comprobados, para que usted genere sus aplicaciones de acuerdo con sus necesidades profesionales o de diversión.

En el Capítulo 1, presentamos información sobre la plataforma Android, así como su historia, arquitectura, qué es un SDK, el entorno de desarrollo Android Studio que usaremos en este material, y, al final, estudiamos el contenido del archivo AndroidManifest como complemento para todas las aplicaciones de los próximos capítulos.

En el Capítulo 2, hacemos referencia al lenguaje de programación Java, que permitirá crear aplicaciones móviles en Android de manera dinámica. Para esto, debemos orientar toda la programación a la tecnología objetos, pues Android trabaja de esa manera: usando métodos getset y constructores.

En el Capítulo 3, nos adentramos en la plataforma Android para conocer los componentes principales de una aplicación Android. A esta parte debemos regresar cada vez que se experimente problemas en los capítulos posteriores, pues es importante dominar el modo de agregar un recurso, definir correctamente una actividad o crear fragmentos dentro de una aplicación, ya que hay aplicaciones que necesitan de estos elementos para funcionar correctamente.

En el Capítulo 4, definimos el uso de los controles que permiten crear una aplicación Android. En este sentido, iniciamos por la distribución de componentes mediante Layouts y, luego, aprendemos el uso de los controles que los contiene y términos con el uso de las notificaciones que en todo dispositivo móvil aparece.

En el Capítulo 5, se presentan controles avanzados para la mejora de nuestras aplicaciones en Android, tales como el manejo de listas y las barras de herramientas que permiten administrar de mejor manera la información de nuestra aplicación.

En el Capítulo 6, usamos los archivos de texto como un gestor de almacenamiento de información que permitirá tener una idea de cómo se maneja la información por medio de clases File y XML.

En el Capítulo 7, nos introducimos al manejo del gestor de base de dato SQLite y definimos las principales funciones con que cuenta para gestionar la información dentro de una aplicación; asimismo, realizamos un mantenimiento completo de información como listar, agregar, modificar y eliminar información.

En el Capítulo 8, usamos clases que permiten activar los servicios dentro de Android. Estos normalmente trabajan en segundo plano dentro de la aplicación y su uso es de suma importancia.

En el Capítulo 9, usamos el servicio Google Maps, con el cual aprenderemos a crear una clave API para su uso, y crearemos aplicaciones que permitirán mostrar su ubicación en tiempo real.

En el Capítulo 10, nos enfocamos en aplicaciones de tipo multimedia que permitirán gestionar audio y video dentro de una aplicación Android. Esto hará que nuestras aplicaciones sean mucho más dinámicas y divertidas.

Por último, en el Capítulo 11, explicaremos cómo se puede publicar una aplicación Android en la tienda virtual más grande del planeta y haremos referencia a los costos que esto demanda.

CAPÍTULO

1



Plataforma Android

1.1 Introducción al Android

Una de las necesidades básicas del ser humano es la comunicación, ya que a partir de esta podrá escuchar, ser escuchado y principalmente interactuar con otras personas; claro está que, en la actualidad, esta necesidad ha abarcado muchos contextos y uno de ellos es la comunicación móvil.

Una de las empresas pioneras en la comunicación móvil fue Motorola que, en la Segunda Guerra Mundial, creó los *handie talkie* y los *walkie talkie* que permitían la comunicación entre los soldados que iban a pie. La diferencia entre *handie* y *walkie* eran sus dimensiones y capacidad de alcance, además de que diferían en el costo de reparación.

Mientras tanto, en el Perú, a inicios de la década de 1990, solo el 3% de la población contaba con telefonía fija. No se podía habla de Internet, pues solo se sabía que existía como un recurso académico para obtener información en investigaciones; el periódico era uno de los medios más importantes para la difusión de información, aún se enviaban cartas por medio de empresas de correo y, con respecto a los videos o multimedia, solo se podían visualizar por una señal de televisión. Por esos tiempos, surgió la empresa Tele2000, que ofrecía comunicación móvil a todo el Perú mediante equipos telefónicos sumamente costosos. Sin embargo, ello nos ayudó a dar un gran paso en la tecnología.

Actualmente, es posible hablar de teléfonos inteligentes que cubren casi todas las necesidades de un usuario: una comunicación eficaz y rápida; contar con una galería de imágenes tomadas desde la propia cámara; visualizar videos y editarlos; y visualizar contenidos web o mantener comunicación en tiempo real mediante videoconferencia. Lo que antes involucraba tener un aparato para cada necesidad, hoy se concentra en un solo aparato.

Toda esta tecnología se debe en parte a Android que, en la actualidad, se encuentran en la mayoría de dispositivos móviles, relojes, televisores, automóviles y aparatos electrodomésticos, los cuales pueden ser considerados como aparatos inteligentes, es decir, *smart* (ver Figura 1).



Figura 1. Internet de las cosas. Tomado de "The internet of things: A beginner's guide to commercial applications and security risks," por J. Fox, 2017. Recuperado de https://www.ifsecglobal.com/global/the-internet-of-things-a-beginners-guide-to-commercial-applications-and-security-risks/

1.2 Historia de Android

Android es un sistema operativo de código abierto para todo tipo de dispositivo móvil. Es considerado líder en el mercado, porque logra cubrir todas las necesidades de los usuarios. Además, es el más completo y funcional sistema operativo móvil, de fácil manejo y principalmente personalizable, con el cual se aprovechan las ventajas tecnológicas que nos ofrecen los aparatos móviles.

Cuenta con una tienda de aplicaciones llamada Play Store. Desde esta se pueden descargar miles de aplicaciones entre gratuitas y pagadas, las cuales convierten a nuestro dispositivo móvil en un elemento muy importante para el ser humano.

En un estudio realizado entre los dos sistemas operativos que revolucionaron el mercado tecnológico, se obtuvo que Android supera en 0.2% a Microsoft, lo cual marca un hecho importante en la historia tecnológica (ver Figura 2).

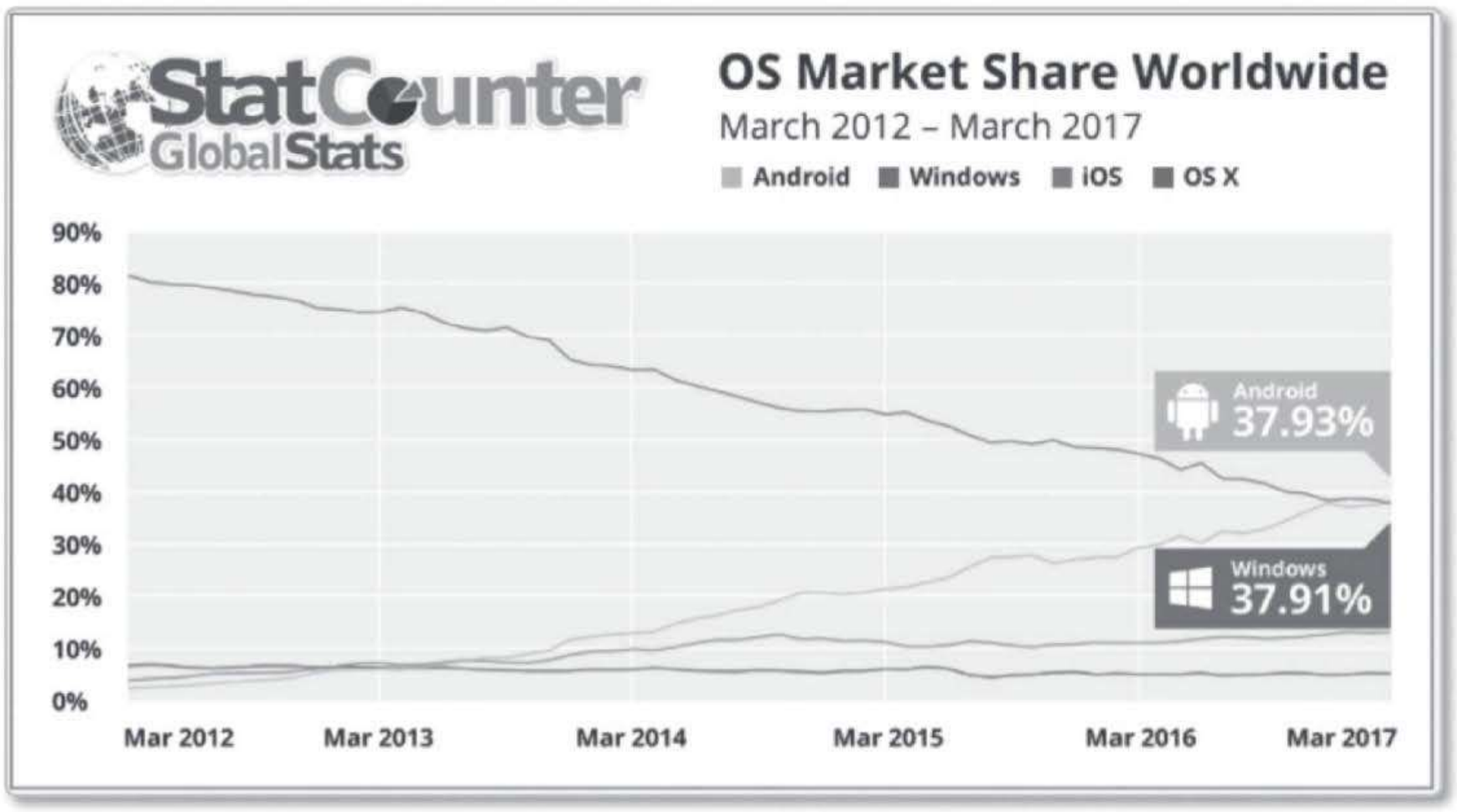


Figura 2. Estadísticas de los sistemas operativos para dispositivos móviles . Tomado de "Android overtakes Windows for first time," por R. Simpson, 2017. Recuperado de http://gs.statcounter.com/press/android-overtakes-windows-for-first-time

Los primeros sistemas operativos eran difíciles de programar, porque se enfocaban mucho en el hardware, es decir, el programador debía comprender perfectamente esta dimensión; en la actualidad, estos son abstraídos mediante el uso de librerías. De este modo, Android cumple con las mismas características que presentan los sistemas operativos, con la diferencia de que se ejecutarán en una plataforma móvil.

La primera aparición de Android ocurrió con la versión oficial 1.1, la cual se publicó en febrero de 2009. Desde entonces, han aparecido diferentes versiones de este sistema operativo. Una característica distintiva es que cada versión del sistema operativo recibe un nombre inspirado en la repostería y cumple un orden alfabético con respecto al resto de versiones, por ejemplo, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo y Pie.

1.3 Open Handset Alliance TM

Es una alianza para dispositivos móviles abiertos. Está constituida por un grupo de empresas de tecnología móvil tanto en hardware como en software, operadoras de telefonía y distribuidores a nivel mundial.

Fue fundada el 5 de noviembre del 2007 como una alianza comercial de 84 compañías con un mismo objetivo: desarrollar estándares abiertos para todo tipo de dispositivo móvil. El mismo día de la fundación de la alianza, se lanzó al mercado tecnológico Android, el cual presenta un sistema operativo de código libre, basado en el núcleo del sistema Linux. Veamos una lista de las empresas que representa el consorcio:

Operadores de telefonía. Se considera los siguientes:

- China Mobile
- KDDI Corporation
- NTT DoCoMo
- Sprint Nextel
- T-Mobile
- Telecom Italia
- Telefónica
- Vodafone
- Softbank
- China Unicom
- China Telecommunications Corporation
- Telus
- Bouygues Telecom

Empresas de software. Se considera las que siguen:

- Ascender Corporation
- eBay
- Esmertec
- Google
- LivingImage
- Myriad
- NMS Communications
- Nuance Communications
- PacketVideo
- SkyPop

- SONiVOX
- SVOX
- NXP Software
- Access
- VisualOn
- Cooliris
- MOTOYA Co., Ltd.
- OMRON

Empresas comercializadoras. Se considera las siguientes:

- Aplix
- Noser Engineering
- The Astonishing Tribe
- Wind River Systems
- Borqs
- Omron Software
- Teleca
- Sasken Communication Technologies Limited
- Accenture
- L&T Infotech
- SQLStar International Inc.
- Wipro Technologies

Empresas fabricantes de semiconductores. Son las siguientes:

- Audience
- Broadcom Corporation
- Intel Corporation
- Marvell Technology Group
- Nvidia Corporation
- Qualcomm
- SiRF Technology Holdings
- Synaptics
- Texas Instruments

- AKM Semiconductor
- ARM
- Atheros Communications
- EMP
- MIPS Technologies
- MediaTek
- Cypress Semiconductor Corporation
- Freescale Semiconductor
- Gemalto
- Renesas Electronics Corporation
- VIA Technologies

Empresas fabricantes de dispositivos. Son las que siguen:

- HTC Corporation
- LG
- Sony
- Motorola (Ahora Motorola Mobility)
- Samsung Electronics
- ASUS
- Garmin
- Huawei
- Sony Mobile Communications (Como Sony Ericsson)
- Toshiba
- Dell
- Hacer
- ZTE
- Alcatel Mobile Phones
- Compal Communications
- Foxconn
- Haier
- Kyocera
- Lenovo Mobile Communication Technology Ltd.
- NEC

- Sharp Corporation
- Saygus Corporation

1.4 Estructura Android

Android, como sistema operativo móvil, se encuentra organizado por capas, las cuales trabajan bajo un mismo núcleo llamado Kernel, basado en Linux (ver Figura 3).



Figura 3. Estructura Android.

Capa 1: Aplicaciones. Es un conjunto de programas que interactúan con el usuario con la finalidad de darle solución a un determinado proceso. Se cuenta, por ejemplo, con aplicaciones nativas, las cuales son desarrolladas para un tipo de dispositivo como los calendarios, la biblioteca de imágenes, las pantallas de inicio, el marcador de teléfono, el lector de correos electrónicos, etc. Por otro lado, también se cuenta con aplicaciones añadidas, que son desarrolladas por empresas especializadas que permiten dar una solución al usuario, por ejemplo, un sistema de control de pedidos en un restaurant o un retocador de imágenes.



Capa 2: Estructura de aplicaciones. Representa un conjunto de herramientas de desarrollo que las aplicaciones deberán usar dentro del sistema Android, ya sean las propias del dispositivo, desarrolladas por terceras compañías, o incluso las que el usuario puede construir. Todas sin excepción usan el mismo conjunto de API.

- Gestor de actividades: Es el encargado de controlar el ciclo de vida de cualquier tipo de aplicación que se ejecute en un dispositivo móvil.
- Gestor de ventanas: Es el encargado de gestionar las ventanas bajo el sistema operativo Android, el cual permite administrar las ventanas que se muestran a los usuarios al hacer uso de cualquier tipo de aplicación en su dispositivo móvil.
- Proveedores de contenido: Es el encargado de encapsular datos o información que puedan ser compartidos entre diferentes aplicaciones. Estas se encapsulan en una lógica interna llamada API, la cual brinda un conjunto de servicios a las aplicaciones que lo soliciten. Por ejemplo, hay aplicaciones que usan la información de los contactos registrados en nuestro dispositivo móvil.
- Gestor de telefonía: Es el encargado de controlar todo lo referente a la telefonía del dispositivo móvil siempre y cuando este tenga el servicio de telefonía activo. Es así que existen tablets con chip para móvil, y, en estos casos, también se administra la gestión de telefonía.
- Gestor de paquetes: Es el encargado de administrar las carpetas con que cuenta el sistema operativo móvil. De este modo, en un dispositivo móvil nuevo, encontramos una estructura de carpetas predeterminada.
- Gestor de visualización: Es en el encargado de gestionar la visualización de las diferentes aplicaciones en el dispositivo móvil.
- Gestor de notificaciones: Es el encargado de controlar las notificaciones que realizan las diferentes aplicaciones en un determinado dispositivo móvil, por ejemplo, los mensajes de texto, las alertas, etc.
- Gestor de recursos: Es el encargado de la gestión de los recursos que podrían solicitar las diferentes aplicaciones desde el dispositivo móvil. Se le llama recurso a cualquier elemento dentro de una aplicación que no sea código fuente.
- Gestor de sensores: Es el encargado de gestionar los sensores dependiendo de la actividad que se realiza en el móvil. Por ejemplo, todos los teléfonos cuentan con un sensor de proximidad, el cual apaga la pantalla mientras usted acerca el teléfono a su oído.
- Gestor de ubicación: Es el encargado de determinar la ubicación geográfica del dispositivo móvil mediante el GPS del equipo.

Capa 3: Bibliotecas. También son llamadas librerías y, junto al núcleo, forman el corazón de Android. Están escritas en lenguaje nativo como C y C++, y son usadas por el sistema para la implementación de aplicaciones que usen hilos, memoria, audio y video, gráficos 3D, base de datos, visualización de páginas web, etc.

Capa 4: Núcleo de Kernel. Es la capa para todo el sistema Android, que provee servicios esenciales para el dispositivo móvil como la administración de energía, el liberador de memoria, la administración de la red Wifi, los controladores de la cámara, etc.

1.5 Características de Android

Contar con Android en algún dispositivo móvil es como llevar una computadora portátil; por tanto, las características son muy similares a las de los sistemas operativos de plataforma. De acuerdo con ello, se menciona las siguientes:

- Código abierto, el cual permite al desarrollador utilizar todo el potencial Android.
- Soporte para el protocolo Wifi 802.11mc.
- Diseño para las notificaciones y ajustes rápidos.
- Núcleo basado en el Kernel de Linux, el cual permite adaptarse a cualquier tipo de dispositivo móvil sin necesidad de contar con un alto volumen de memoria interna.
- Permite la adaptabilidad a infinidad de pantallas y resoluciones.
- Usa SQLite para la administración y gestión de la información.
- Posee diferentes modalidades de mensajería.
- Incluye un acceso a desinstalar aplicaciones desde la pantalla principal del dispositivo.
- Navegador web basado en WebKit, usado para Google Chrome y Safari de Apple.
- Soporte para el desarrollo Java con su máquina virtual.
- Soporte para formatos de imágenes, video y audio.
- Posee un diseño de modo de ahorro de energía.
- Soporte para HTML, HTML5, Adobe Flash Player, etc.
- Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software.
- Posee un modo de no molestar con acceso rápido.
- Posee capturas de pantalla mediante la combinación de teclas volumen y encendido del aparato.
- Gestión para doble cámara de algunos dispositivos.
- Editor de imágenes desde las pantallas capturadas.
- Soporte para Bluetooth de hasta cinco dispositivos conectados a la vez.
- Google Talk para realizar videollamadas.
- Multitarea real de aplicaciones.

1.6 Arquitectura Android (Librerías)

Android basa su arquitectura en el modelo Kernel de Linux, el cual le permite posicionarse en miles de dispositivos móviles.

1.6.1 Zygote

Es un proceso primario para todas las aplicaciones Android. Contiene bibliotecas del sistema y los frameworks usados por casi todas las aplicaciones que se ejecutan en dispositivos móviles. Este proceso funciona como una plantilla para cada nueva aplicación.

1.6.2 Davilk Virtual Machine

Considerado como la máquina virtual para Android, la cual permite a los desarrolladores crear aplicaciones Java que se puedan ejecutar en el entorno móvil. Cumple la misma función que realiza la máquina virtual de Java, la cual permite obtener los códigos de bytes obtenidos a partir de la compilación y ejecutarlos en cualquier dispositivo móvil. Entre las principales características que presenta la máquina virtual Dalvik tenemos:

- Cada aplicación que se ejecuta en un dispositivo móvil creará un proceso independiente con una instancia exclusiva; es decir, la aplicación no consumirá recursos del dispositivo; solo lo necesario para ejecutarse.
- Su arquitectura está basada en registros, porque está demostrado que estas son mucho más rápidas que la administración de procesos por medio de pilas. Los registros, al ser texto plano, son ejecutados rápidamente; en cambio, las pilas usan la memoria del aparato y lo tornan lento.

Asimismo, realiza la gestión de los procesos convirtiendo los archivos compilados en archivos .class. Luego, estos son transformados en archivos .dex, y finalmente son comprimidos en archivo APK que se ejecutan en cualquier dispositivo móvil.

1.6.3 ART – Android RunTime

Es conocido como Android ART y considerado como el sucesor de la máquina virtual Dalvik. Presenta notables mejoras en la ejecución de procesos como la incorporación del "Just in-Time", que permite compilar el código solo cuando se usa la aplicación. Asimismo, crea un archivo de compilación adicional que permite abrir la aplicación más rápido, pues ya no se tiene que compilar. Con esto, se mejora también el rendimiento de la batería del dispositivo, pues el procesador trabaja solo lo necesario. Art reemplazo completamente a Dalvik a partir de la versión Lollipop de Android.

1.6.4 File System

Es el encargado de administrar y gestionar la distribución de archivos y registros. Asimismo, optimiza el espacio libre que pueda tener el dispositivo. Android cuenta con dos sistemas: EXT4 y F2FS. El primero presenta características como la creación de 64000 carpetas y es capaz de mover elementos de hasta 16TB, mientras que el segundo está diseñado para dispositivos de almacenaje externo.

1.7 Android SDK

Es conocido como Software Development Kit de Android y contiene un conjunto de herramientas de desarrollo, lo cual le permite crear e implementar aplicaciones Android desde el diseño, así como la depuración, la ejecución y la puesta en un emulador (ver Figura 4).

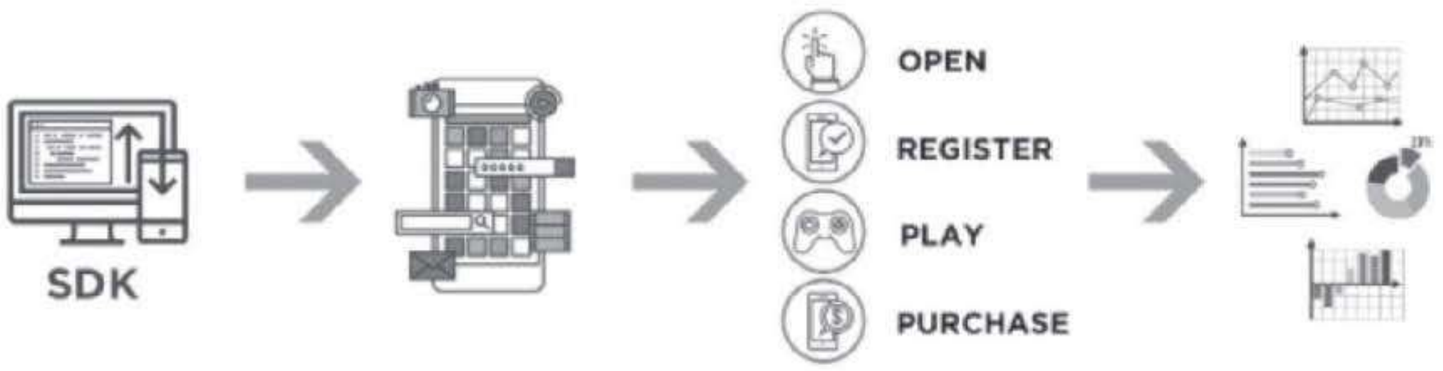


Figura 4. Funcionalidad del sistema operativo Android. Tomado de "How to make your Mobile Application GDPR-Ready," por S. Paneva, 2018. Recuperado de https://blog.sashido.io/how-to-make-your-mobile-application-gdpr-ready/

Contiene los siguientes elementos:

- Depurador de código
- Librerías o bibliotecas
- Simulador para emular las aplicaciones
- Documentación
- Ejemplos de código

1.8 Tipos de procesos de Android

Android gestiona los procesos solicitados por el usuario administrando de manera adecuada la memoria, pues esta necesita liberarse de los procesos pasados y mantenerse ligera. En este sentido, existe una jerarquía de procesos para un buen desempeño del dispositivo. La jerarquía de importancia tiene cinco niveles:

- Proceso de primer plano: Se le llama así a la aplicación que el usuario se encuentra usando en un determinado momento. Es necesario que él mismo libere memoria cerrando las aplicaciones de primer plano antes de empezar con otro proceso.
- Proceso visible: Son aquellos que permiten que el proceso de primer plano se encuentre en dicho nivel. Un error en este proceso podrá afectar lo que el usuario visualiza en primer plano.
- Proceso de servicio: Son procesos que se mantienen en ejecución al margen de lo que ocurra en primero plano. Podemos mencionar, por ejemplo, la descarga de información o la reproducción de música en segundo plano.
- Proceso de segundo plano: Son procesos que no son visibles directamente por el usuario y que pueden ser cerrados por el sistema sin afectar el primer plano. Normalmente, hay muchos procesos corriendo en el segundo plano que son almacenados en una lista llamada LRU (menos usado recientemente). Un ejemplo sobre el proceso de segundo plano es el reloj o la reproducción de sonidos en el dispositivo.
- Procesos vacíos: Son procesos de espera que permiten ejecutar aplicaciones mucho más rápidas a partir de la segunda ejecución. Estos generan un equilibrio entre los recursos del dispositivo y las memorias.

1.9 Versiones Android

En la Tabla 1, se sintetizan las versiones.

Tabla 1. Versiones oficiales Android

NOMBRE COMERCIAL	VERSIÓN	AÑO DE LANZAMIENTO	DESCRIPCIÓN
Apple Pie	1.0	SET-2008	El primer dispositivo móvil en incorporarlo fue el HTC Dream. Era también conocido como Google Phone. También, incluía la primera versión de la tienda virtual llamada Android Market.
Banana Bread	1.1	FEB-2009	Se presentó en dispositivos como el T-Mobile G1, el cual incorporó algunas mejoras importantes con respecto a la versión inicial.
CupCake	1.5	ABR-2009	Introduce el teclado virtual y la posibilidad de agregar algunas widgets.
Donut	1.6	SET-2009	Produce una mejora en los cuadros de búsqueda y renueva el aspecto inicial que presentaba la tienda virtual.
Eclair	2.1	OCT-2009	Integra los contactos del dispositivo móvil con las principales redes sociales como Facebook o Twitter.
Froyo	2.2	MAY-2010	Destaca la velocidad de todo su sistema al ejecutar las aplicaciones y realizar navegaciones web. Además, incluye la posibilidad de convertir al dispositivo móvil en un hotspot.
GingerBread	2.3	DIC-2010	Renueva la interfaz de usuario con mejoras en su resolución y agrega a su teclado virtual la función de corregir textos mientras se escribe.
HoneyComb	3.0	FEB-2011	Renueva la interfaz del usuario incorporando una barra en la parte superior con acceso rápido a notificaciones y otras características del sistema.
Ice Cream Sándwich	4.0	OCT-2011	Los widgets se separan de las aplicaciones y se muestran independientes. También, incluye el reconocimiento facial de los usuarios como tema de seguridad.
Jelly Bean	4.1	JUL-2012	Supone una mejora en las notificaciones al usuario e incorpora la aplicación Google Now como una característica importante de esta versión.
KitKat	4.4	OCT-2013	Introduce el tema de reducción de sus elementos, lo cual mejora considerablemente el aspecto. Asimismo, implementa la funcionalidad de la aplicación Hangouts como un servicio oficial de Google.
Lollipop	5.1.1	NOV-2014	Destaca la forma de funcionamiento de las notificaciones y el consumo de la batería. Modifica la máquina virtual Dalvik por ART (Android Runtime).
Marshmallow	6.0.1	OCT-2015	Incorpora un mejor rendimiento y estabilidad en el sistema operativo. Añade un soporte de huellas dactilares.

NOMBRE COMERCIAL	VERSIÓN	AÑO DE LANZAMIENTO	DESCRIPCIÓN
Nougat	7.0	JUN-2016	Incorpora el manejo de varias aplicaciones en un mismo entorno. De este modo, se puede abrir dos o más ventanas y distribuirlas de manera adecuada. Asimismo, mejora la duración de la batería. También, permite personalizar la barra de navegación en los dispositivos. Ahora se podrá responder desde las notificaciones. Finalmente, implica una mejora en el explorador de archivos.
Oreo	8.0	AGO-2017	Incorpora el ahorro de energía limitando los procesos en segundo plano. Además, mejora considerablemente la mensajería por medio de notificaciones; mejora el diseño incorporando líneas guías para el establecimiento de íconos en la pantalla, y amplia el soporte para las redes Wifi.
Pie	9.0	AGO-2018	Incorpora herramientas Digital Wellbeing, que permiten tener información sobre el uso del dispositivo. Permite la navegación. Agrega imágenes o *stickers* en las notificaciones. Amplía la gestión de seguridad y el rendimiento del dispositivo móvil.

1.10 Entorno de desarrollo

1.10.1 Android Studio

Es un entorno de desarrollo integrado que permite crear aplicaciones móviles para Android bajo la licencia Apache 2.0, la cual lo califica como una aplicación gratuita de desarrollo. Entre las principales características que presenta Android Studio se considera el entorno claro y robusto. Cuenta con una facilidad para depurar aplicaciones móviles y testearlas en cualquier dispositivo móvil. Asimismo, incluye un editor de código y plantillas que permite ayudar en el desarrollo de aplicaciones.

Descarga e instalación. Android Studio está disponible para entornos como Windows, MacOs y GNU/Linux. Así es que, cuando lo descargue, considere el tipo de sistema operativo.

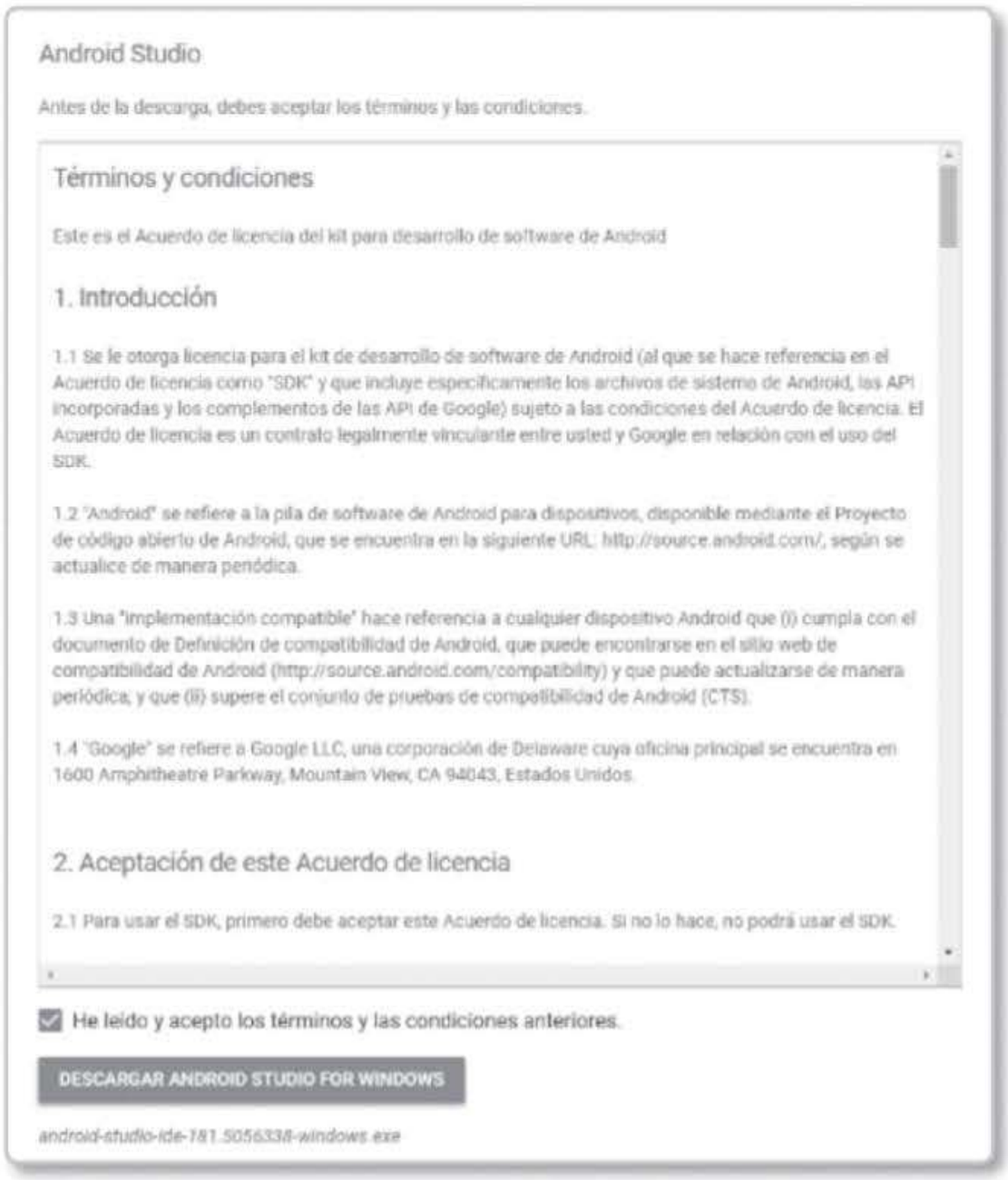
URL de descarga: **https://developer.android.com/studio/?hl=es-419**

A continuación, mostraremos el proceso de instalación de Android Studio:

Para nuestro caso, seleccionaremos la descarga de Android Studio para Windows de 64 bits, tal como se muestra en la siguiente imagen:



Luego, debemos activar el check de "He leído y acepto los términos y las condiciones anteriores" y seleccionamos el botón "Descargar Android Studio for Windows".



La siguiente imagen muestra información relacionada con Android Studio, mientras procede a la descarga de la aplicación.



Luego, desde la carpeta **Descargas**, observamos la aplicación que permitirá instalar Android Studio.



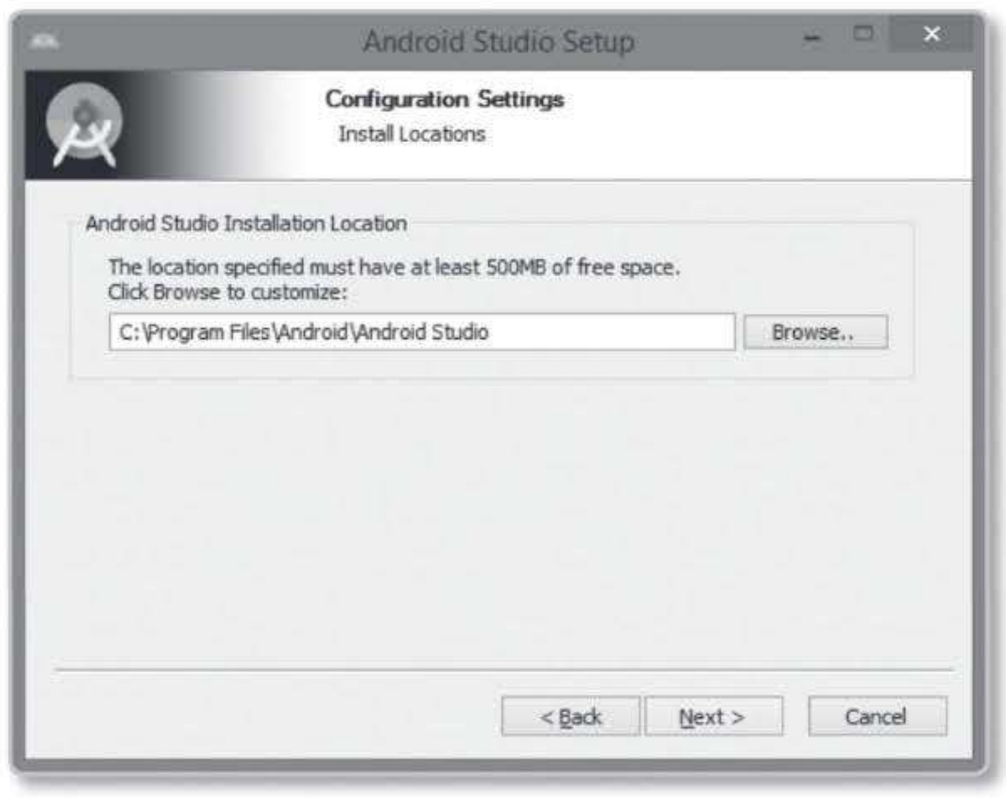
En el proceso de instalación, presionamos el botón **Next** para continuar.



Luego, verificamos que se encuentren activadas las opciones **Android Studio** y **Android Virtual Device**, y presionamos el botón **Next**.



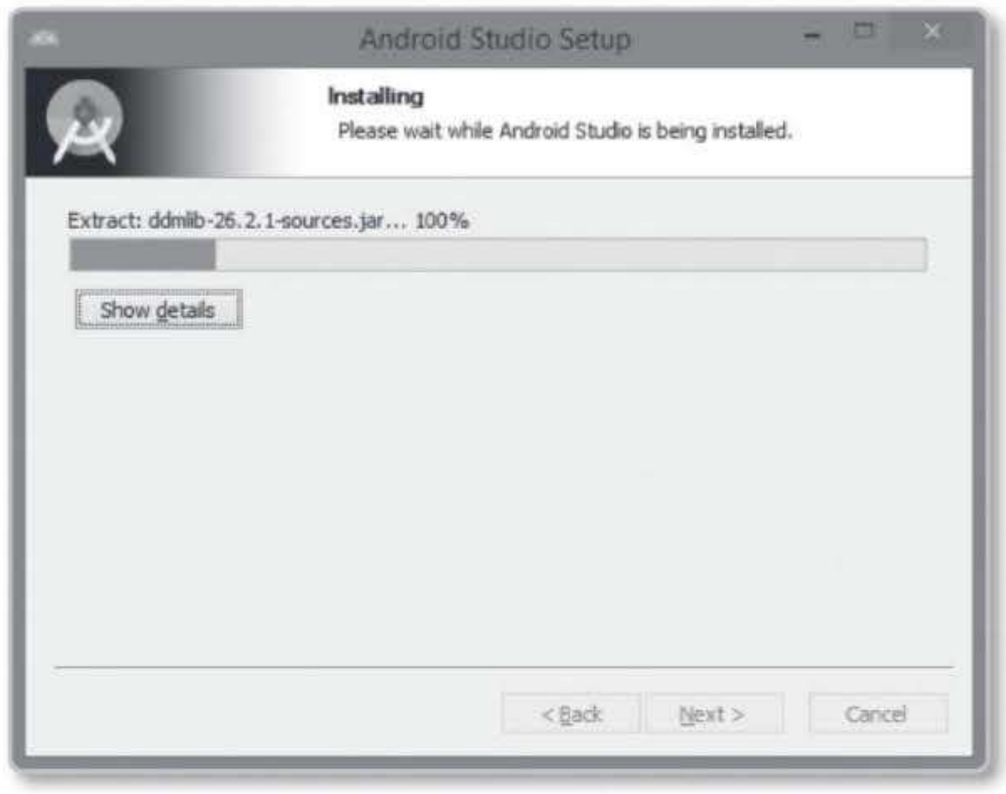
Luego, debemos seleccionar una ubicación para la instalación. Por defecto, se instalará dentro de la carpeta **Android** de **Program Files**.



Luego, presionamos el botón **Install** para iniciar el proceso de instalación



Luego, verificamos que la instalación se ejecute correctamente.



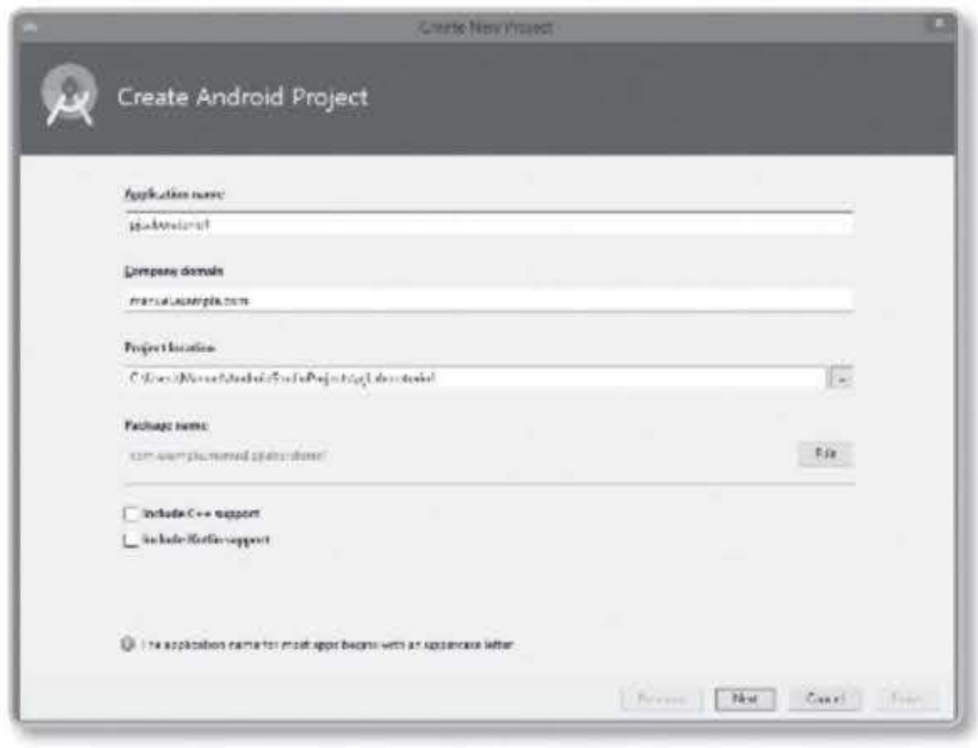
Finalmente, cuando se muestre la ventana de **Completing Android Studio Setup**, presionemos el botón **Finish** para terminar con el proceso de instalación y abrir el entorno Android Studio.



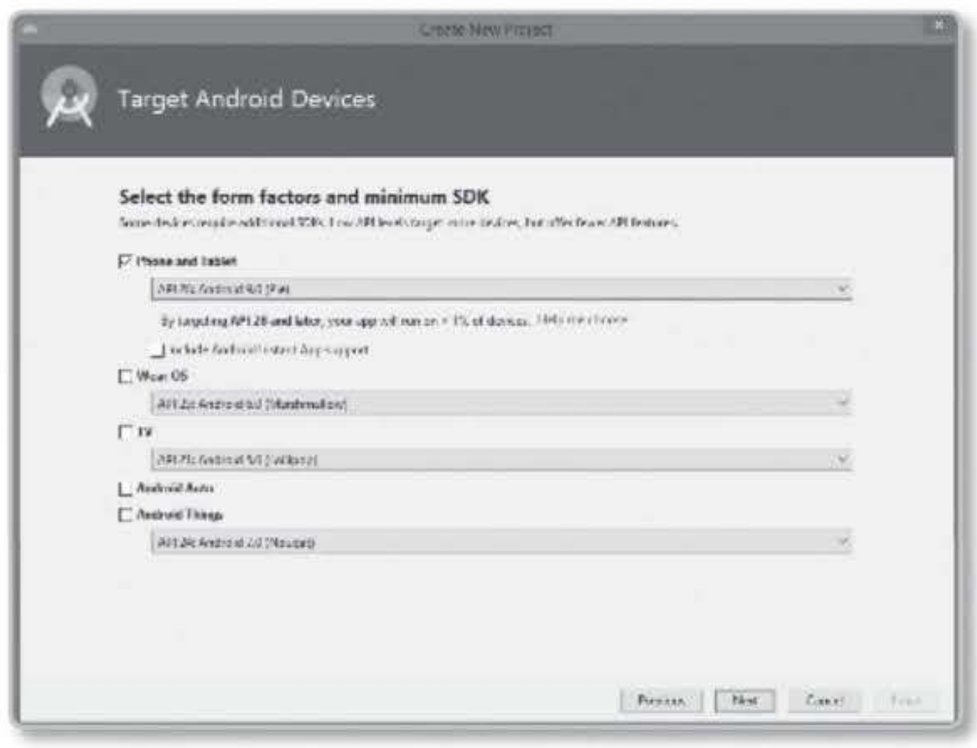
Creando un proyecto con IDE Android Studio. Al iniciar la aplicación Android Studio, seleccionemos **Start a new Android Studio project**.



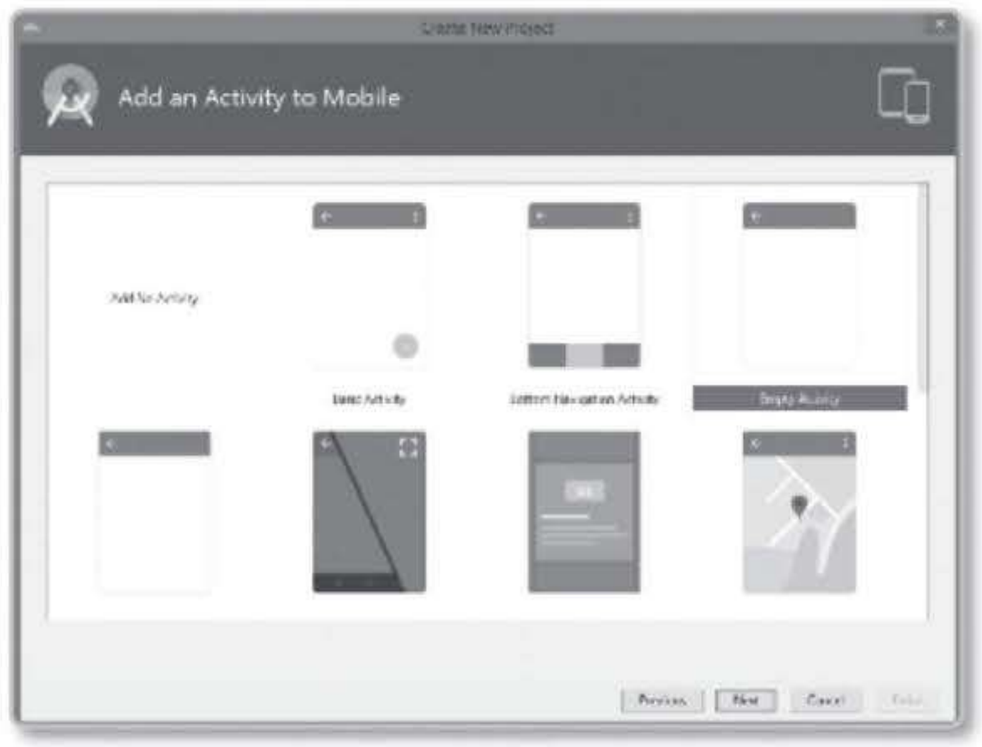
Luego, debemos asignarle un nombre a la aplicación. En nuestro caso, hemos colocado **pjLaboratorio1**, con lo cual hacemos referencia a nuestro primer proyecto. Opcionalmente, se puede colocar el nombre de la compañía en la cual se está creando la aplicación y, finalmente, se podría especificar la ubicación del proyecto desde **Project Location**.



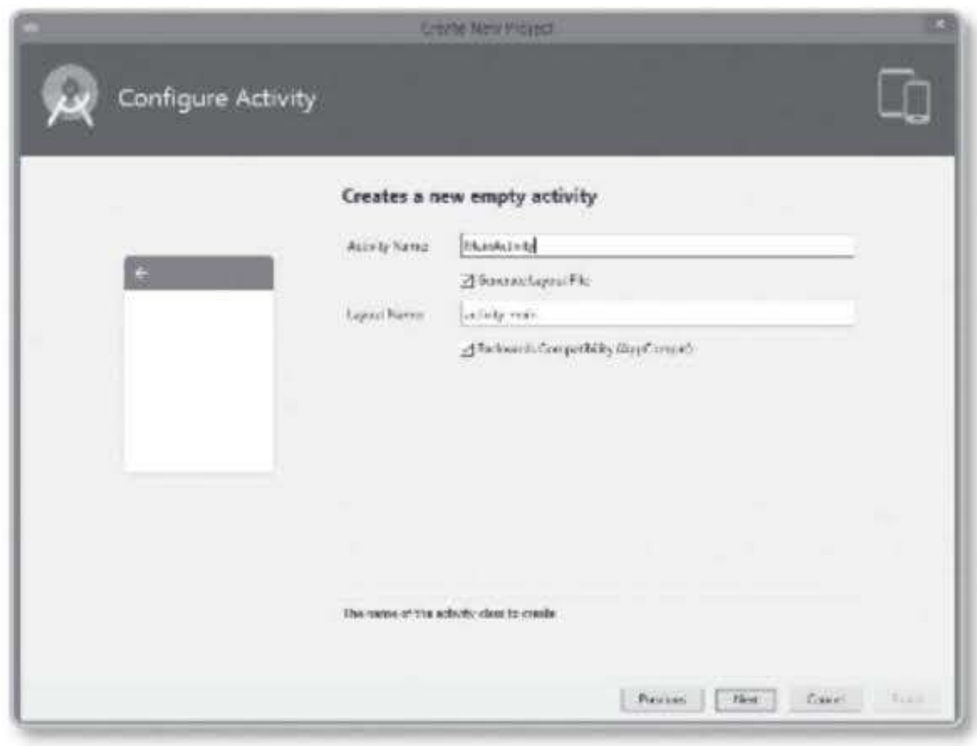
Luego, seleccionamos la plataforma donde se ejecutará la aplicación Android. En nuestro caso, hemos seleccionado **API 28 Android 9.0** para Android Pie o más conocido como Android P.



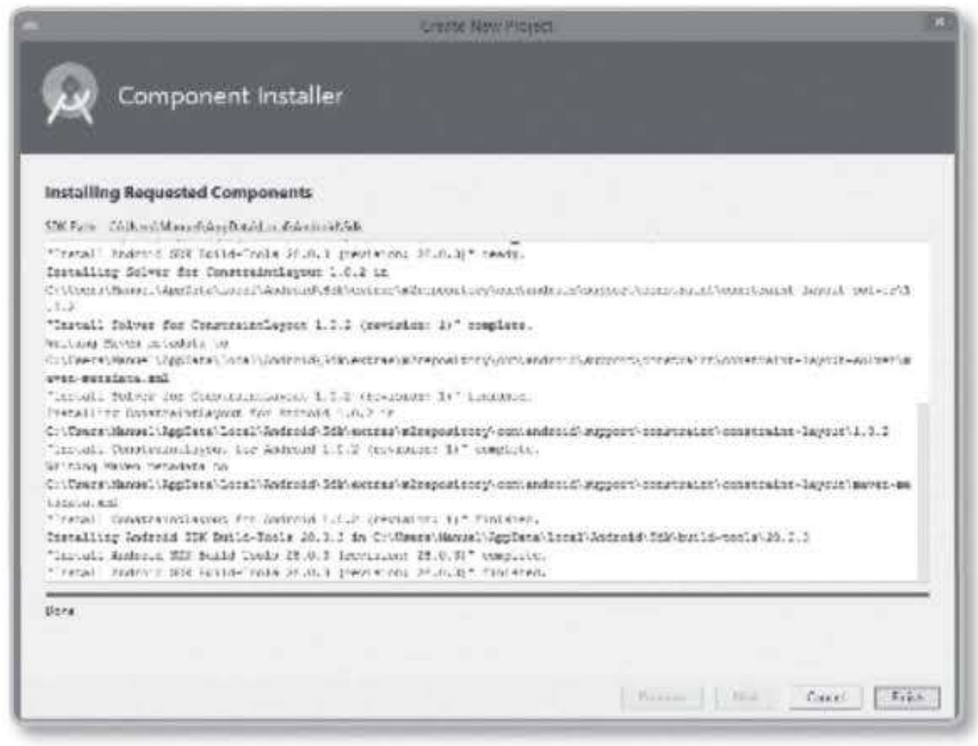
Seguidamente, seleccionamos la plantilla de trabajo, a partir de la cual comenzaremos con el diseño de la aplicación. En este caso, seleccionaremos **Empty Activity**.



Luego, asignaremos un nombre a nuestra actividad. Comúnmente, la primera actividad se suele llamar MainActivity, y a partir de la segunda se recomienda modificar el término "Main" por el contenido real de la actividad, por ejemplo, LoginActivity si fuera un activity para usuario y clave.



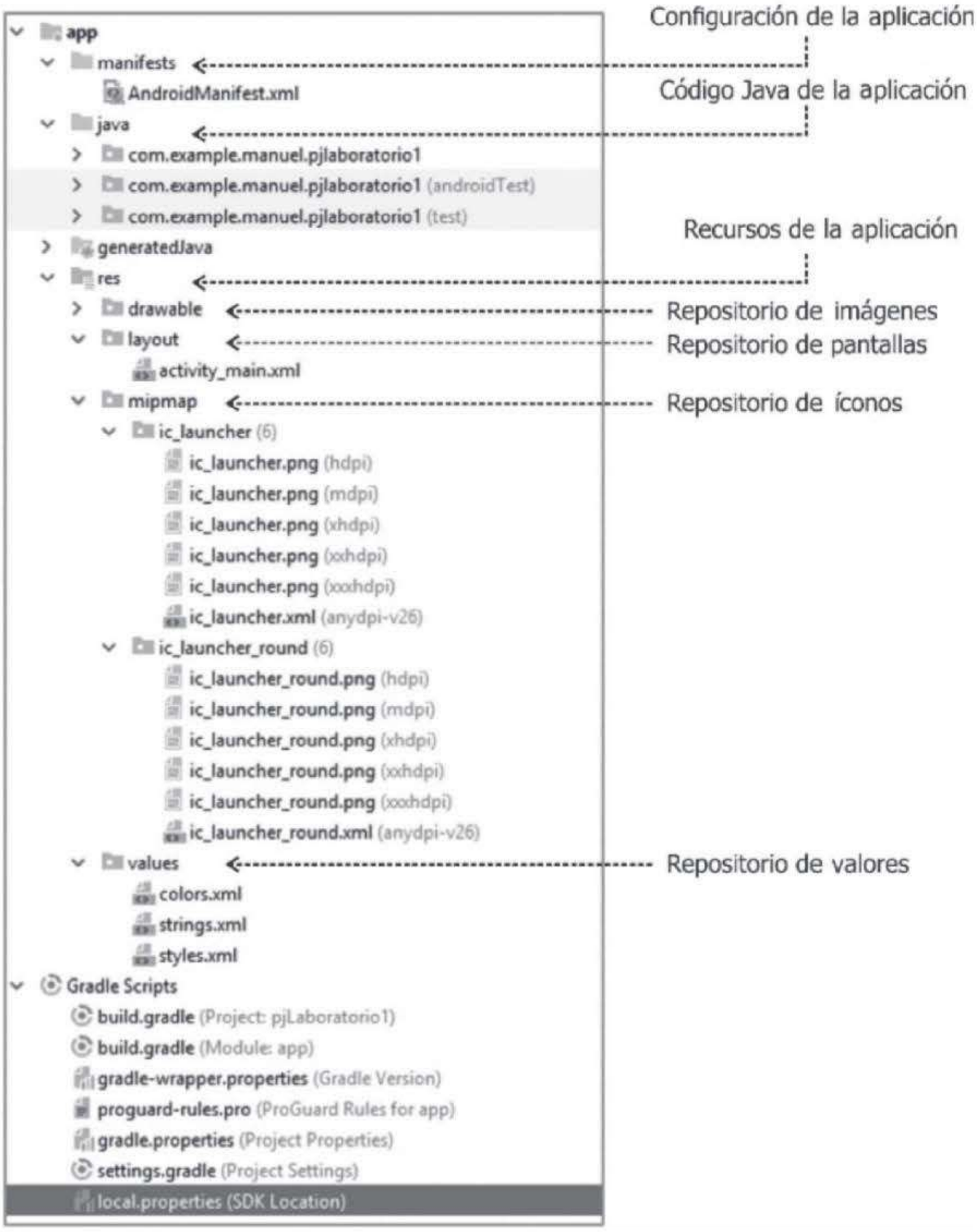
Finalmente, Android Studio instalará los componentes necesarios para la aplicaciones y podremos continuar con el desarrollo de nuestra aplicación. Para esto, debemos seleccionar el botón **Finish**.



1.11 Fundamentos de aplicaciones Android

1.11.1 Estructura de un proyecto en Android Studio

A continuación, se presenta la estructura del proyecto.



- Carpeta manifest: Contiene el archivo **AndroidManifest.xml**, desde el cual muestra información sobre la aplicación que el sistema Android debe tener en cuenta para ejecutarse.
- Carpeta java: Contiene archivos de extensión .java, cuyo código fuente se puede ver con solo presionar doble clic. En esta sección, podremos colocar nuestros códigos, que permitirán tener interacción con el usuario final.
- Carpeta res: Contiene todos los archivos considerados como recursos para la aplicación. Desde aquí, podremos administrar las imágenes, videos, activitys, colores que sean necesarios en una aplicación. Estos, a su vez, están organizados en:

- Drawable: Contiene todos los recursos gráficos de la aplicación.
- Mipmap: Contiene todos los íconos que se pueden usar en la aplicación.
- Layout: Contiene todas las pantallas que se visualizarán en el proyecto.
- Values: Contiene declaración de colores, dimensiones o textos usados en una aplicación.

1.11.2 Definición de Activity

Activity android es un elemento contenedor dentro de una aplicación, el cual representa la interfaz del usuario. Es a partir de aquí donde el usuario podrá interactuar con una aplicación Android. Comúnmente, una actividad ocupa toda la pantalla del dispositivo. Si no es así, entonces se encuentra flotando. Por otro lado, cuando inicia una actividad, la actual se detiene, y se genera una pila de actividades que debe ser cerrada por el usuario a menos que el programador haya temporadizado la permanencia de la actividad (ver Figura 5).

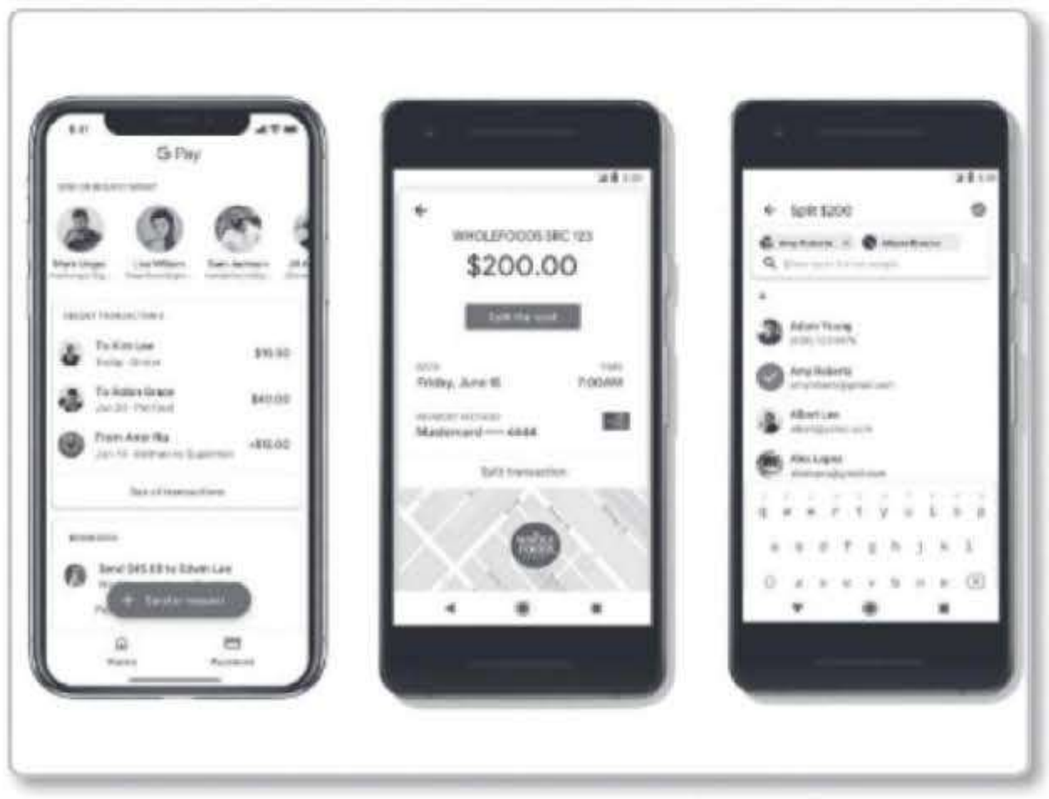


Figura 5. Actividades en Android. Tomado de "Time Timer® Android App," por Time Timer, s.f. Recuperado de https://www.timetimer.com/products/time-timer-android-app.

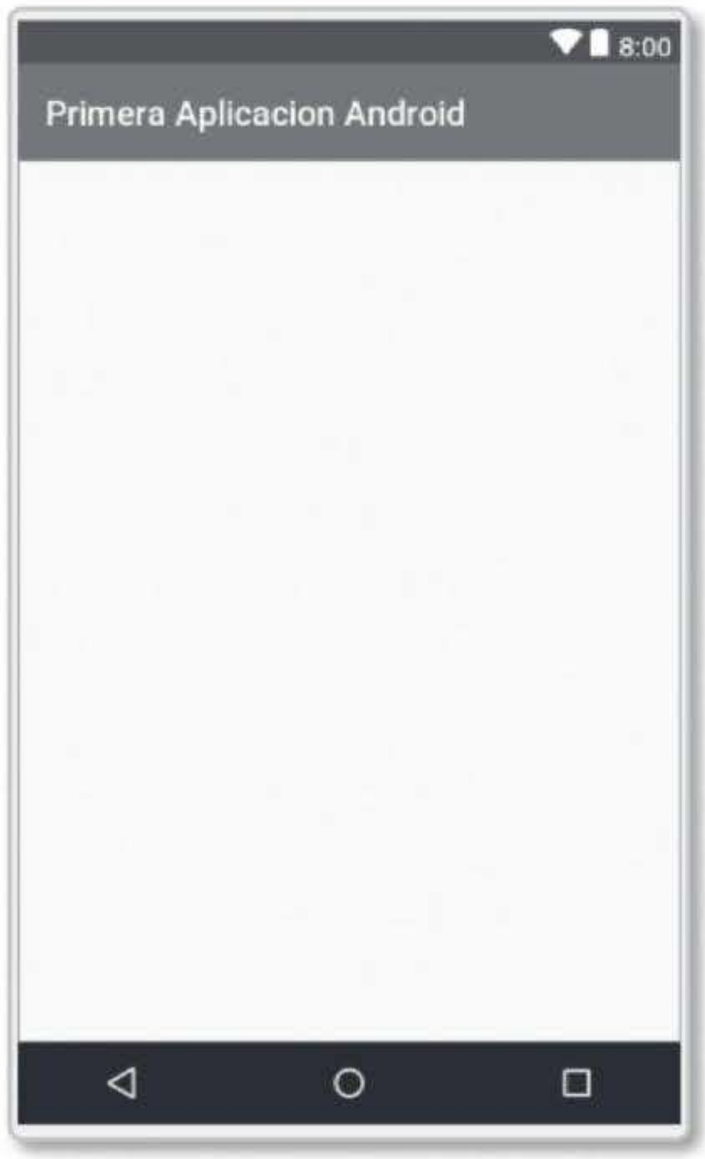
Cada activity consta de dos partes:

- Parte lógica: Contiene un código que permite gestionar el contenido de la actividad así como responder a eventos que el usuario puede solicitar. Finalmente, este archivo tiene la extensión .java.

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

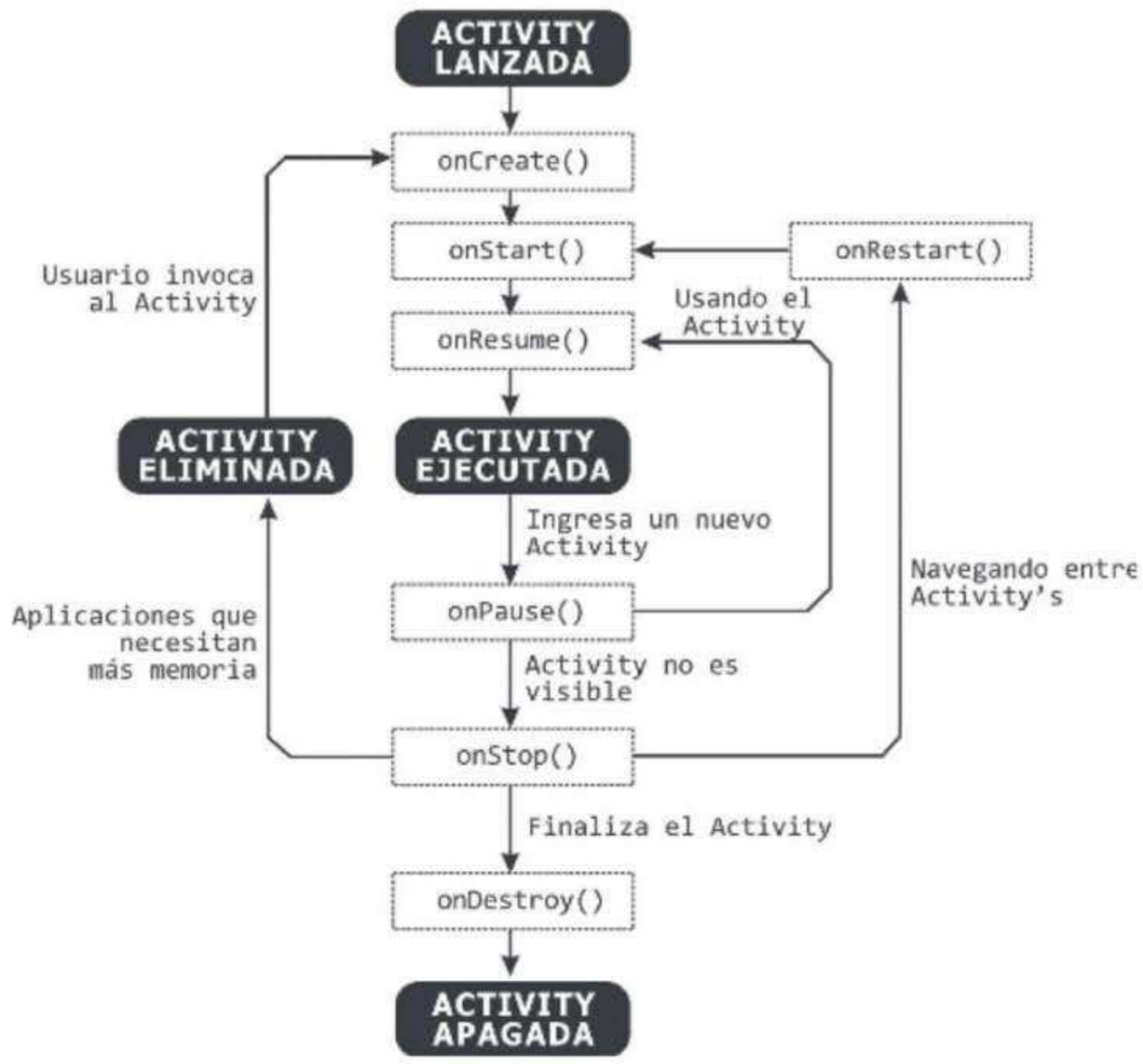
</android.support.constraint.ConstraintLayout>
```

- Parte gráfica: Contiene etiquetas XML que permiten visualizar la interfaz de una aplicación al usuario. Finalmente, este archivo tiene la extensión .xml.



1.11.3 Ciclo de vida de un activity

Todas las actividades tienen el mismo ciclo de vida. Es a partir de aquí que debemos pensar en el desarrollo de aplicaciones, pues normalmente se implementan navegaciones entre actividades, y especialmente debemos anticipar cuál será el accionar del usuario final de la aplicación.



En la Tabla 2, se muestra los principales métodos del ciclo de vida de un activity.

Tabla 2. Métodos del ciclo de vida de un activity

MÉTODO	DESCRIPCIÓN
onCreate	Método que se activa al ejecutar un activity por primera vez. Desde aquí podemos especificar qué se quiere ver al iniciar la actividad. El método que se ejecuta después de su llamada es onStart.
onRestart	Método que se activa justo antes de volver a iniciar el activity. El método que se ejecuta después de su llamada es onStart.
onStart	Método que se activa antes de mostrar el activity al usuario. Aquí se puede tomar dos caminos: si la actividad se oculta, se invoca al método onStop, y si la actividad es usada por el usuario, se invoca al método onResume.
onResume	Método que se activa antes de iniciar la interacción con el usuario; podría ser que la actividad se encontrase en la pila de actividades. El método que se ejecuta después de su llamada es onPause.
onPause	Método que se activa cuando el sistema reanuda una nueva actividad. Aquí se puede tomar dos caminos: si la actividad se oculta, se invoca al método onStop, y si esta es empleada por el usuario, se invoca al método onResume.
onStop	Método que se activa cuando la actividad ya no es visible para el usuario. Aquí se puede tomar dos caminos: si la actividad desaparece, se invoca al método onDestroy, y si esta es empleada por el usuario, se invoca al método onRestart.
onDestroy	Método que se activa antes de finalizar el ciclo de vida de la actividad. Literalmente la actividad se destruye y, para volver a invocarlo, se deberá iniciar desde el método onCreate.

1.11.4 Agregar un activity

En una aplicación, se pueden tener varias interfaces dentro del mismo proyecto; es decir, debe agregarse a este más activitys. Originalmente, un proyecto nuevo siempre viene con un activity por defecto llamado **mainActivity**.

Pasos:

- Presionemos clic derecho sobre la carpeta **Layout > New > Activity > Empty Activity**
- Asignemos un nombre a la actividad **SplashActivity**, tal como se muestra en la siguiente imagen:

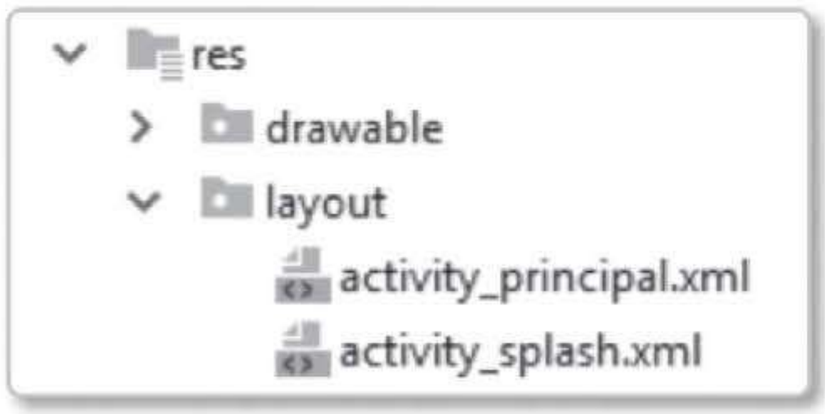


1.11.5 Eliminar un activity

Debemos tener en cuenta que, al eliminar un activity, también estamos eliminando todos los elementos contenidos en este.

Pasos:

- Visión desde la lista de actividades mostrada desde la carpeta **layout**:



- Presionemos clic derecho sobre la actividad que se desea eliminar y seleccionemos la opción **Delete...**

1.11.6 Modificar la actividad de inicio del proyecto

Cuando se ejecuta una aplicación, una de las actividades recibe la acción de presentarse al inicio. Esto se debe a un código de programación que permite indicar quién debe mostrarse inicialmente.

Si tenemos una aplicación que cuenta con dos actividades llamadas **PrincipalActivity** y **SplashActivity**, y queremos que cada una de ellas sea la actividad de inicio, debemos colocar los siguientes códigos:

Actividad Principal como inicio de la aplicación:

```
<activity android:name='.PrincipalActivity'>
    <intent-filter>
        <action android:name='android.intent.action.MAIN' />
        <category android:name='android.intent.category.LAUNCHER' />
    </intent-filter>
</activity>

<activity android:name='.SplashActivity'> </activity>
```

Actividad Splash como inicio de la aplicación:

```
<activity android:name='.PrincipalActivity'> </activity>

<activity android:name='.SplashActivity'>
    <intent-filter>
        <action android:name='android.intent.action.MAIN' />
        <category android:name='android.intent.category.LAUNCHER' />
    </intent-filter>
</activity>
```

1.11.7 Modificar el tamaño y tipo de letra del editor en Android Studio

- **File > Settings**...
- Seleccionemos **Editor > Font**
- Modifiquemos el tipo de letra y el tamaño, tal como se muestra en la siguiente imagen:



Se recomienda usar la fuente **Consolas** y, con respecto al tamaño, este se establecerá de acuerdo con sus necesidades.

1.11.8 Uso de los recursos

El desarrollo de una aplicación Android promueve un conjunto de elementos como textos, imágenes, videos, íconos y otros componentes. Se recomienda tener todos estos elementos en archivos ordenados. Así mismo, todos los recursos se encuentran organizados en la carpeta **res** de la aplicación.

AGREGAR IMÁGENES

- Seleccionamos los archivos tipo imágenes. Hay que considerar que los archivos tipo .png no tienen fondo.
- Copiamos dichos archivos.
- Los pegamos en la carpeta **drawable**. Seleccionamos una ubicación y presionamos la tecla **OK**.
- Luego, podremos usar las imágenes haciendo referencia a la carpeta **drawable**.

AGREGAR INTERFACES O LAYOUTS

- Presionamos clic derecho sobre la carpeta **layout**.
- Seleccionamos **New > Activity > Empty Activity**.
- Asignamos un nombre a la actividad.
- Presionamos la tecla **Finish**.

AGREGAR ÍCONOS

- Presionamos clic derecho sobre la carpeta **mipmap**.
- Seleccionamos **New > Image Asset**.
- Seleccionamos una imagen preferentemente de tipo .png desde la opción **path**.
- Modificamos algunas opciones, verificamos la vista previa del ícono y presionamos **Next**.
- Presionamos el botón **Finish** y se creará una carpeta con el nombre del ícono, dentro de la cual habrá varios archivos de tipo .png, listos para ser usados en la aplicación.
- Luego, podremos usar los íconos haciendo referencia a la carpeta **mipmap**.

AGREGAR COLORES

- Presionamos doble clic sobre el archivo **colors.xml**, que se encuentra dentro de la carpeta **values** de los recursos.
- Agregamos el siguiente código dentro de las etiquetas **<resources></resources>** :

```
<color name='nombreColor'>#CodigoColor</color>
```

- Luego, podremos usar los colores haciendo referencia con el término **@color/colorSeleccionado**.
- Podemos visitar el siguiente URL: **https://www.rapidtables.com/web/color/RGB_Color.html**, para encontrar el tono de color y su código hexadecimal.

AGREGAR CADENAS DE TEXTOS

- Presionamos doble clic sobre el archivo **strings.xml** que se encuentra dentro de la carpeta values de los recursos.
- Agregamos el siguiente código dentro de las etiquetas **<resources></resources>**:

```
<string name='nombreTexto'>Contenido Textual</string>
```

- Luego, podremos usar el texto haciendo referencia con el término **@string/nombreTexto**.

AGREGAR ESTILOS

Se puede definir estilos con recursos previamente implementados como es el caso de los colores o las cadenas de texto.

- Presionamos doble clic sobre el archivo **styles.xml** que se encuentra dentro de la carpeta **values** de los recursos.
- Agregamos el siguiente código dentro de las etiquetas **<resources></resources>**:

```
<style name='NombreEstilo' parent='DefinicionModeloTema'>
    <item name='nombreColor'>@color/colorSeleccionado</item>
</style>
```

- Luego, podremos usar el estilo haciendo referencia con el término **@style/nombreEstilo**.

1.12 Estudio del archivo Android Manifest

El proyecto de Android contiene el archivo **AndroidManifest de tipo XML**, desde el cual se puede configurar los componentes propios del proyecto, así como especificar la actividad de inicio o cambiar el título del proyecto, entre otras configuraciones.

Veamos el script del archivo AndroidManifest:

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

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Configuración Inicial de la aplicación

Configuración de la actividad de inicio

El archivo AndroidManifest se encuentra dentro de la carpeta **manifest** y, dentro de la configuración inicial de la aplicación, presenta las siguientes opciones:

- **Version**: Indica el número de versión actual de la aplicación. Se asume que cada versión es una actualización de esta.
- **Encodign**: Determina la representación de caracteres especiales como tildes o ñ mediante una secuencia de caracteres. En otras palabras, especifica el sistema de caracteres vigente. Normalmente, se coloca utf-8 ,el cual cuenta con los caracteres estándar según la ISO 10646.
- **Package**: Especifica la ruta de la aplicación, la cual se especificó al crearla.
- **Icon**: Especifica el modelo de ícono que se presentará al momento de ejecutarlo en un dispositivo móvil. Debemos tener en cuenta que, para modificar su valor, esta se debe realizar desde los recursos.
- **Label**: Especifica el título que se mostrará en toda la aplicación. Debemos considerar que, para modificar su valor, esta se debe realizar desde los recursos.
- **Theme**: Especifica un conjunto de características propias de la aplicación como el color de barra y otras especificaciones. Debemos tener en cuenta que, para modificar su valor, esta se debe realizar desde los recursos.

CASO DESARROLLADO 01: Interfaz inicial con Android Manifest

Se debe implementar una aplicación que permita modificar el título, y especificar un ícono de la aplicación desde Android Studio.

Se realizan los siguientes pasos:

1. Creamos un nuevo proyecto desde Android Studio (**File > New > New Project...**) y le asignamos un nombre, tal como se muestra en la siguiente imagen:



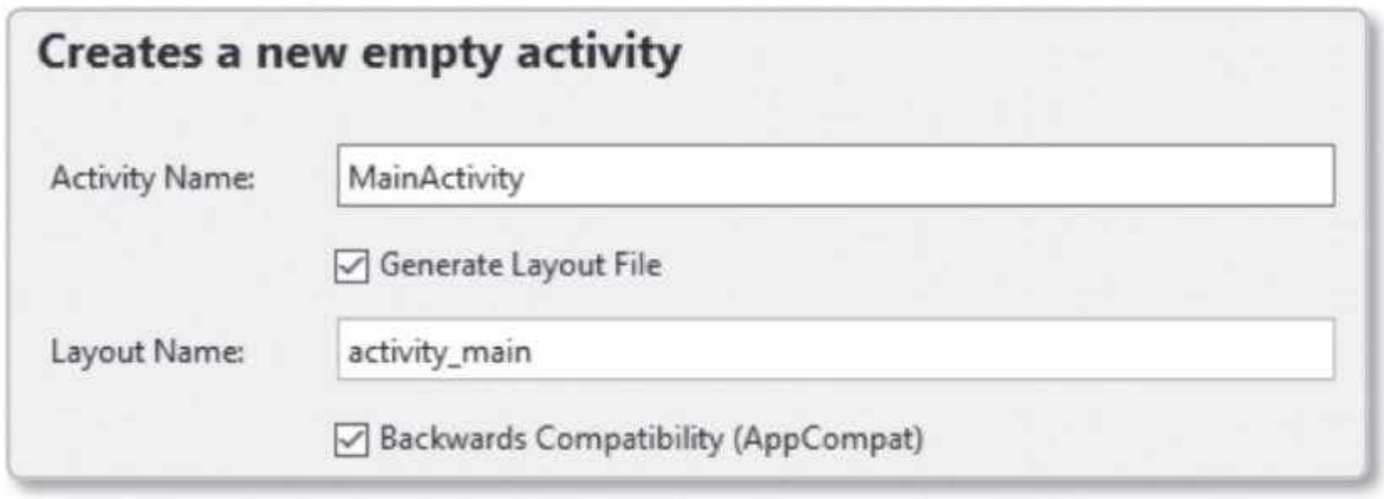
2. Seleccionamos el destino de la aplicación **Phone and Tablet** y la versión Android 6.0, porque, según las estadísticas, el 62.6% de dispositivos móviles cuenta con Marshmallow.



3. Seleccionamos la plantilla de actividad **Empty Activity**.



4. Luego, asignamos un nombre a la actividad. Hay que tener en cuenta que debemos activar **Generate Layout File** para generar el archivo .java de la actividad, ya que desde aquí podemos darle dinámica al proyecto.



ASIGNANDO UN ÍCONO

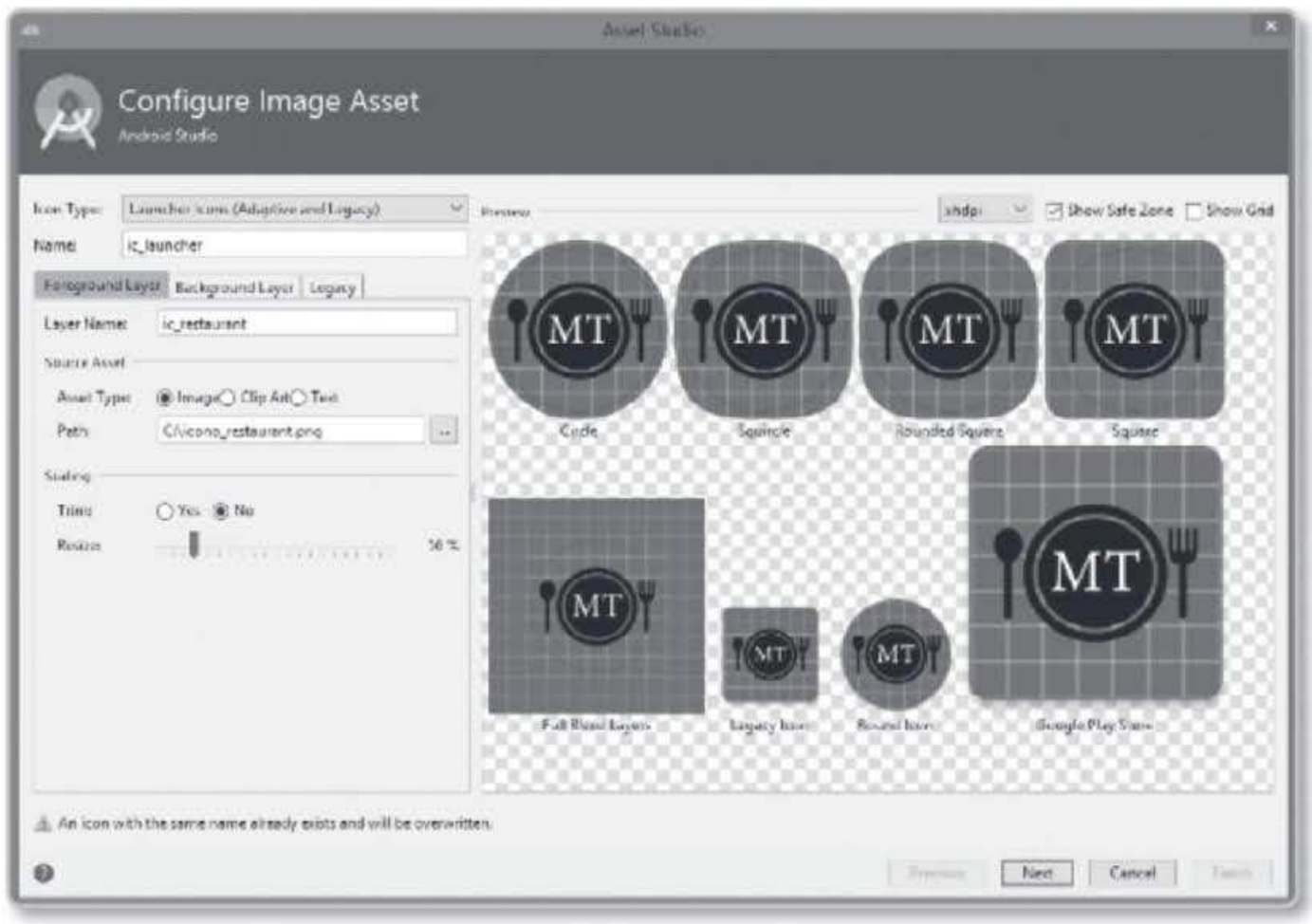
Debemos tener en cuenta que el ícono asignado solo se mostrará al acceder a la aplicación:

5. Para asignar un ícono a nuestra aplicación, debemos contar con una imagen de tipo .png. En nuestro caso, usaremos el siguiente modelo:

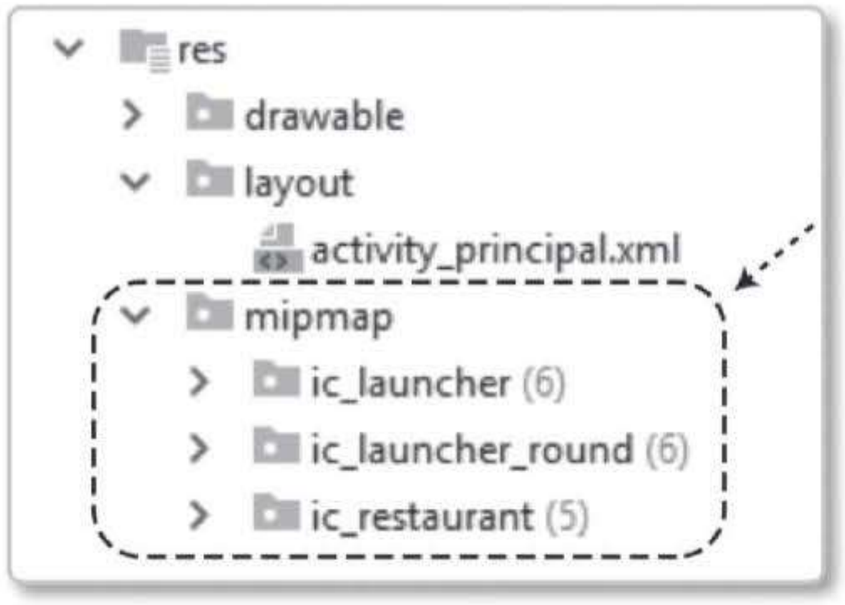


icono_restaurant.png

6. Luego, desde Android Studio seleccionamos **File > New > Image Asset**; asimismo, desde la opción Path, seleccionamos la imagen que se asignará como ícono. Finalmente, seleccionamos **Next** y **Finish**:



7. Veremos que la carpeta **res** cuenta con una subcarpeta llamada **ic_restaurant (5)**, tal como se muestra en la siguiente imagen:



8. Luego, dentro del archivo **AndroidManifest**, modificamos las siguientes líneas de sentencia:

```
android:icon="@mipmap/ic_restaurant"
android:roundIcon="@mipmap/ic_restaurant"
```

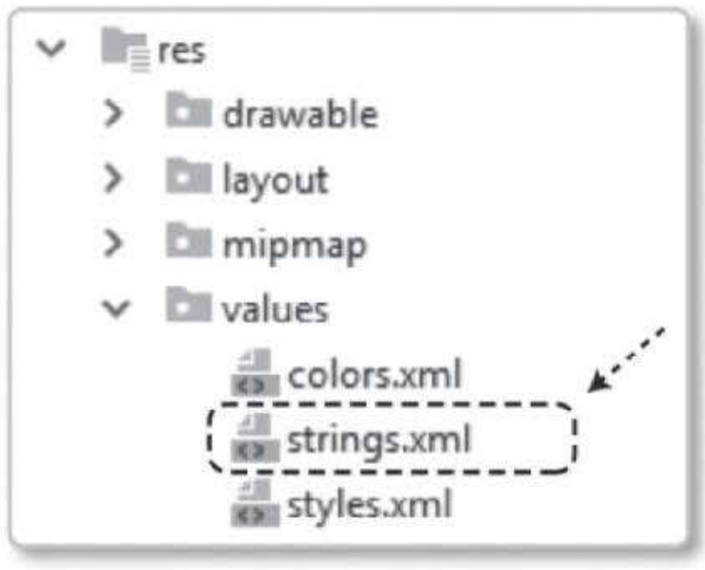
CAMBIANDO EL COLOR DE LA PANTALLA DE LA ACTIVIDAD

9. Para el cambio de color de fondo, entramos al archivo **activity_main.xml** y cambiamos la propiedad **background**, tal como se aprecia en la siguiente imagen:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    android:background="@color/colorPrimaryDark">
</android.support.constraint.ConstraintLayout>
```

CAMBIANDO EL TÍTULO DEL PROYECTO

10. Para el cambio del nombre de la aplicación, entramos al archivo **strings.xml**, tal como se observa en la siguiente imagen:



Luego, modificamos el atributo **name**, tal como se muestra en el siguiente código:

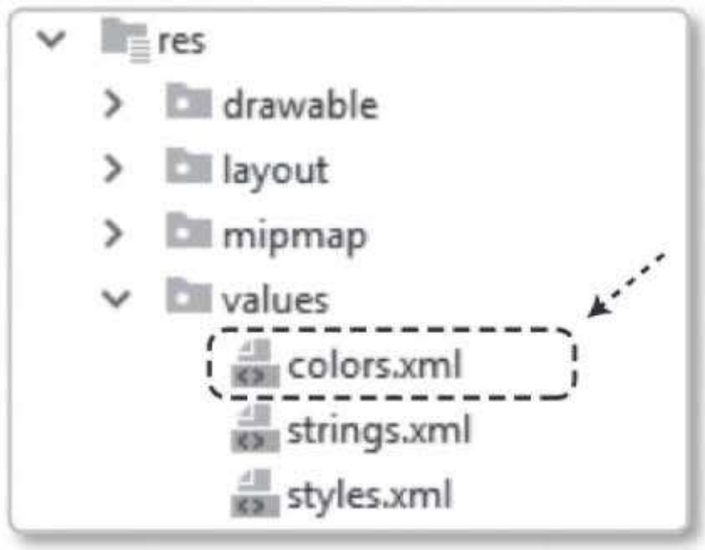
```
<resources>
    <string name="app_name">Sistema de Control de Restaurant</string>
</resources>
```

Otra forma de hacerlo consiste en modificar la propiedad Label del archivo **AndroidManifest**:

```
<application
    android:label="Sistema de Control de Restaurant"
</application>
```

MODIFICAR EL COLOR DE LA BARRA DE TÍTULO DEL PROYECTO

11. Para modificar el color de la barra, entramos al archivo **colors.xml**, tal como se aprecia en la siguiente imagen:



Luego, modificamos el atributo **name=colorPrimary** a **#0099FF** que representa al color azul, tal como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#0099ff</color>
    <color name="colorPrimaryDark">#00ffdc</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

OCULTANDO LA BARRA DE TÍTULO DEL PROYECTO

12. Para ocultar la barra, entramos al archivo **styles.xml**, tal como se observa en la siguiente imagen:



Luego, modificamos el atributo **parent** a **NoActionBar**, tal como se observa en el siguiente código:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

13. Finalmente, el archivo **AndroidManifest.xml** se debe mostrar de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.manuel.pjcapitulo01">

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_restaurant"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_restaurant"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
       <activity android:name=".PrincipalActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

              <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

14. Para ejecutar la aplicación en el emulador, debemos presionar las teclas “Shift+F10” o, desde el menú de opciones, seleccionar **Run > Run app**.

CAPÍTULO

2



Fundamentos Java para Android

CAPÍTULO

2



Fundamentos Java para Android

2.1 Tipos de datos

Representa un conjunto de valores que puede tomar una variable. Al definir un tipo de datos, se define la capacidad que puede tener un valor. Java para Android no cuenta con una variedad de tipos de datos, pero deja la posibilidad de crear sus propios tipos de datos mediante una clase, como es el caso de la clase String. Veamos algunas características de los tipos de datos:

- Se tienen ocho tipos de datos primitivos agrupados en valores enteros, reales, caracteres y boleanos.
- Un tipo de datos determina el rango que una variable puede tener.
- La definición de una variable en Java para Android no distingue la plataforma de trabajo.
- Java para Android es un lenguaje de programación estrictamente tipado; es decir, toda variable debe ser declarada con su propio tipo de datos.

Según el tipo de información que se almacene dentro de una variable, se asignará un tipo de datos, para lo cual se sigue dos tendencias:

- **Tipos de datos primitivos:** Estos representan a un solo valor; también, son llamados escalares. La forma de identificar un tipo de datos primitivo es que siempre se inicia con una letra minúscula.
- **Tipos de datos por referencia:** Estos representan a un conjunto de valores, por ejemplo, los arreglos, las clases, las interfaces, etc. La forma de identificarlos es que siempre se inician con una letra mayúscula.

Veamos los tipos de datos primitivos agrupados por categorías en la Tabla 3:

Tabla 3. Tipos de datos primitivos

Categoría	Tipo de datos
Enteros	byte (entero de 8 bits) short (entero de 16 bits) int (entero de 32 bits) long (entero largo de 64 bits)
Reales	float (coma flotante de precisión simple de 32 bits) double (coma flotante de precisión doble de 64 bits)
Carácter	char (carácter)
Boleanos	boolean (lógico)

Ahora, veamos las capacidades de los tipos de datos primitivos (ver Tabla 4).

Tabla 4. Capacidad de los tipos de datos primitivos

Tipo de datos	Capacidad mínima	Capacidad máxima
byte	-128	+127
short	-32768	+32767
int	-2147483648	+2147483647
long	-9223372036854775808	+9223372036854775807
float	±3.40282347E+38	±1.40239846E-45
double	±1.79769313486231570E+308	±4.94065645841246544E-324
char	\u0000	\uFFFF

También, se debe considerar el valor por defecto de cada tipo de datos primitivos. Esto podría ayudar en la programación cuando se declaran variables globales o de clase, ya que dicha declaración no necesita especificación de valores iniciales. Esto se debe a que se le asigna su valor por defecto. Veamos una lista de dichos valores en la Tabla 5:

Tabla 5. Valor por defecto de los tipos de datos primitivos

Tipo de datos	Valor por defecto
short	0
int	0
long	0L
float	0.0f
byte	0
double	0.0d
char	'\u000'
boolean	False

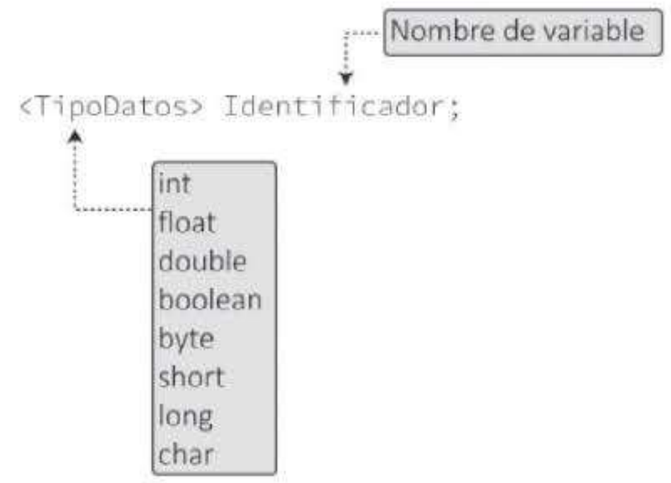
2.2 Variables

Una variable es una posición de memoria destinada a almacenar cierta información expresada en valores. Dicho valor debe ser del mismo tipo con que se declara la variable. Así mismo, toda variable tiene la capacidad de ser modificable durante el ciclo de una aplicación Android.

Java para Android es considerado un lenguaje altamente tipado; es decir, todo lo que use en la programación debe ser declarado usando identificadores. Al declarar variables, se debe tener en cuenta lo siguiente:

- Asignar un nombre dependiendo de lo que se va a almacenar.
- Si la variable declarada está totalmente en mayúsculas, así deberá usarse dentro de la aplicación. Esto se debe a la sensibilidad a las mayúsculas y minúsculas de Java.
- Toda variable se compone de un nombre identificativo, un tipo de datos tipo y un valor que contendrá.
- El tiempo de vida de una variable dependerá del uso que tenga dentro de una clase o método; es decir, al finalizar un método, también finaliza la variable.

Formato:



Veamos algunas especificaciones de declaración de variables en la Tabla 6.

Tabla 6. Valores iniciales de los tipos de datos primitivos

DESCRIPCIÓN	SENTENCIA DE DECLARACIÓN
Declaración de una variable	int n;

Declaración de una variable con inicialización.	int n=0;
	double monto=0;
	String descripcion="";

Declaración de una variable y asignación de valor.	String fechaNacimiento; fechaNacimiento="04/05/1976

2.3 Programación orientada a objetos

La programación orientada a objetos (POO) es una forma especial de implementar aplicaciones, más cercana a cómo expresaríamos las cosas en la vida real. Con POO se tiene que aprender a pensarlas de una manera diferente, es decir, implementar aplicaciones en términos de objetos, propiedades y métodos.

Tradicionalmente, la forma de enfrentar una complejidad del software consistía en usar la programación modulada, en la cual un bloque de código se podía reutilizar solo dentro de la misma aplicación. La programación orientada a objetos tiene el mismo principio; lo que la diferencia de los demás estilos de programación va más allá de la modulación de su código: se dice que tiene como pilares los siguientes elementos:

- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo
- Reutilizar código

En la actualidad, programar orientado a objetos es una actividad común entre los programadores, ya que los lenguajes de programación promueven este estilo. Durante varios años, los desarrolladores se han dedicado a construir aplicaciones muy parecidas que permiten resolver problemas haciendo que el programador sea una persona indispensable en la organización, ya que solo este conoce el código que implementa. La POO se creó con la idea principal de no depender de una persona particular, de manera que otras personas puedan utilizarlas y adelantar su trabajo, y que

consigamos que el código se pueda reutilizar. Por ejemplo, podemos observar que, en un cajero automático, existen muchas funcionalidades y características que se emplean en la POO.



Una de las características que presentan los elementos en la POO es la declaración de la visibilidad frente a los demás objetos. Es decir, podemos definir ciertos alcances que pueden tener dichos elementos, tal como se muestra en la Tabla 7:

Tabla 7. Alcance de las visibilidades

Situación del elemento contenido en el paquete	Visibilidad privada (private)	Visibilidad pública (public)	Visibilidad protegida (protected)	Sin Visibilidad
Dentro de la misma clase.	Sí	Sí	Sí	Sí
Desde una subclase.	No	Sí	Sí	No
Desde una clase implementada dentro del mismo paquete.	No	Sí	Sí	Sí
Desde el exterior de la misma clase.	No	Sí	No	No

2.4 Clase-objeto

Una clase es un elemento básico de la programación orientada a objetos, y representa a una plantilla desde la que se pueden crearlos. Técnicamente, una clase es la descripción de un tipo de objeto en la que se aplica la abstracción de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos sobre un objeto y se le define características y funcionalidades, en realidad lo que estamos haciendo es programar en una clase. La forma de trabajar de una clase se observa en el siguiente diseño (Figura 6).

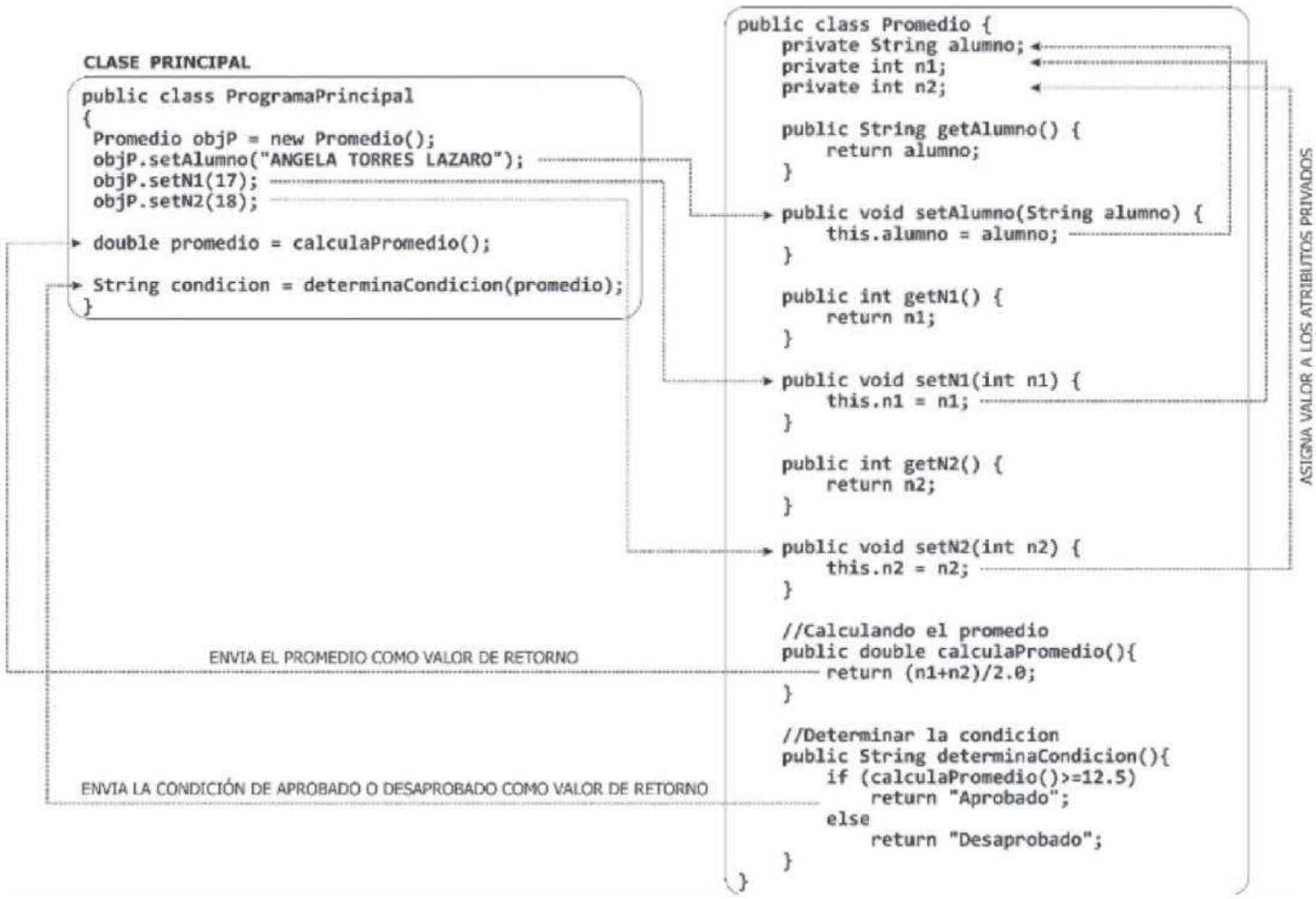
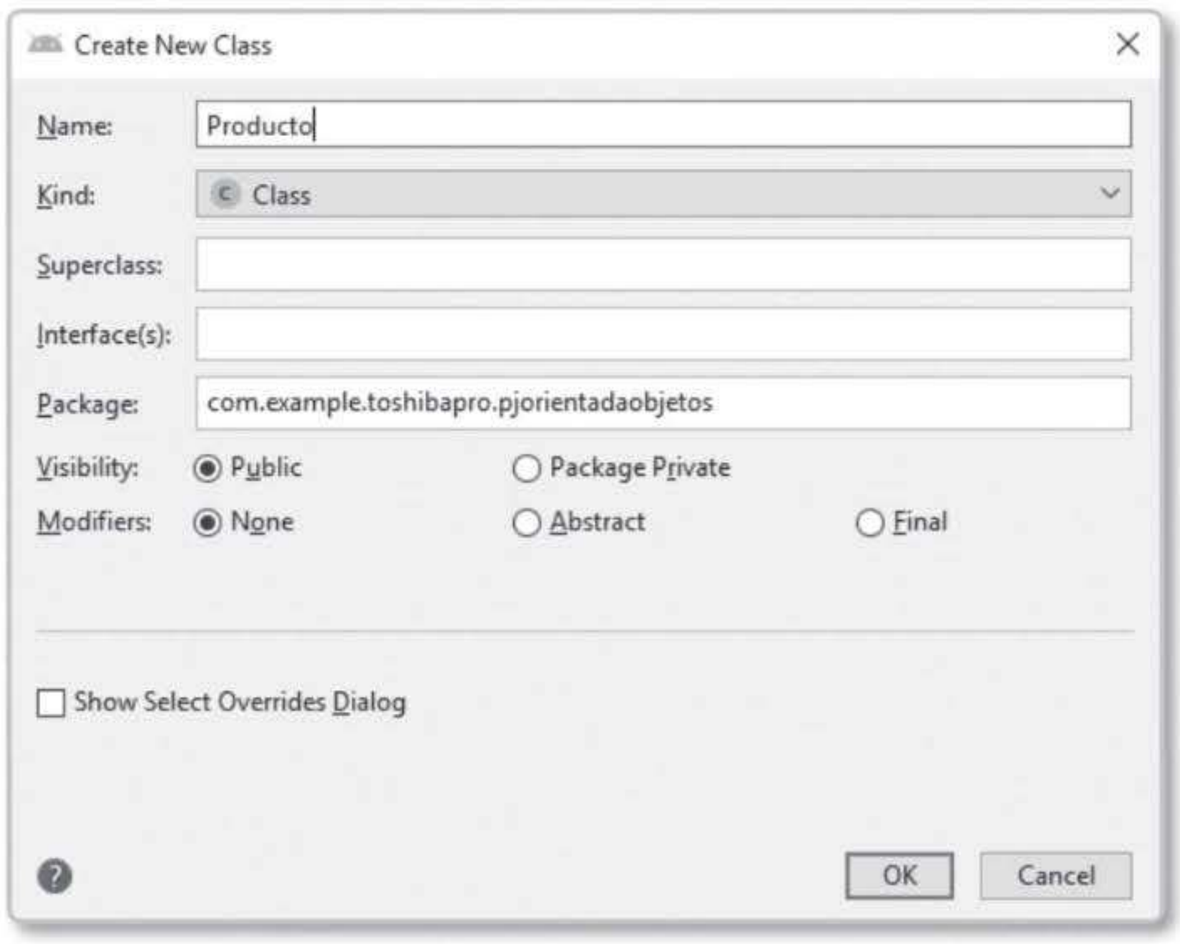


Figura 6. Código orientado a la programación con objetos.

2.4.1 **Crear una clase dentro del proyecto Android:**

La clase debe crearse preferentemente en el mismo paquete del archivo **MainActivity**, realizando los siguientes pasos:

- Hacemos clic derecho sobre la carpeta Java
- Seleccionamos la opción **New > Java Class**.



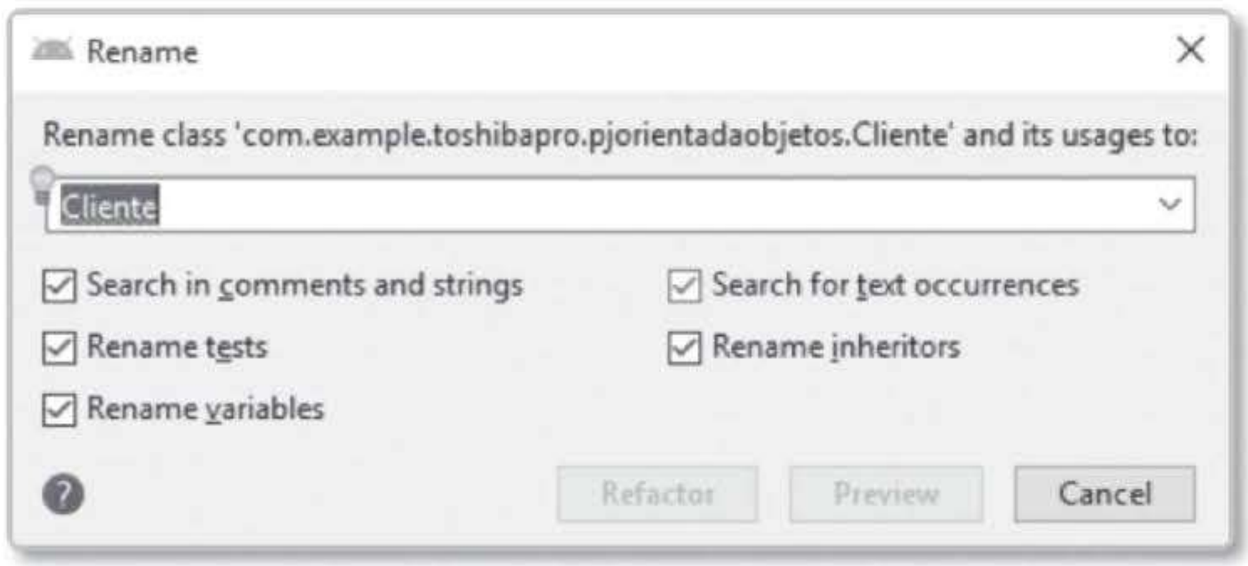
- Colocamos el nombre de la clase y nos aseguramos de que la visibilidad sea pública.
- Inicialmente, la clase se muestra de la siguiente manera:

```
package com.example.pjcapitulo2;
public class Producto {
}
```

2.4.2 Modificar el nombre de la clase

Si por alguna razón necesita modificar el nombre de la clase, se recomienda eliminarla y volverla a crear pero si desea modificar el nombre lo puede realizar de la siguiente manera:

- Hacemos clic derecho sobre la clase.
- Seleccionamos **Refactor > Rename**.

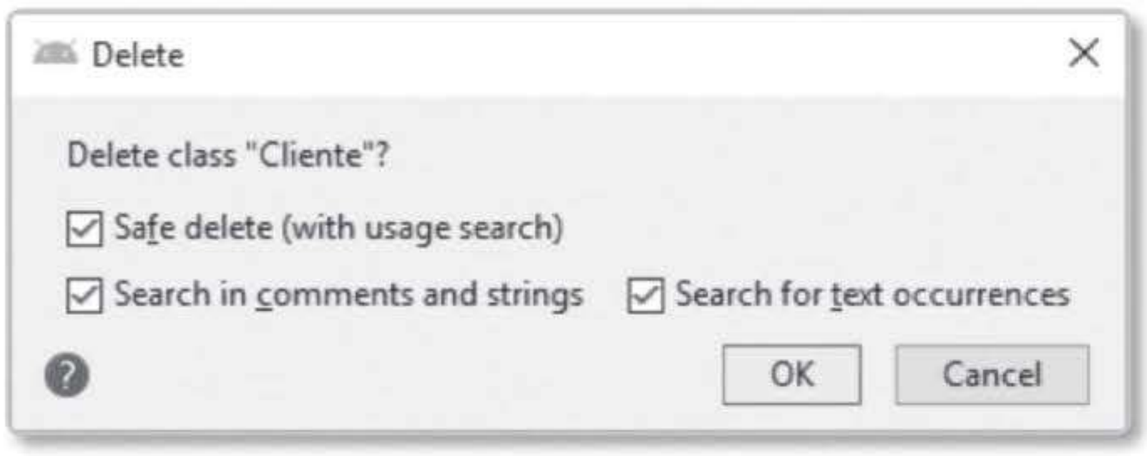


- Modificamos el nombre en la sección **Rename class**.

2.4.3 Eliminar una clase

Para eliminar una clase, se debe realizar los siguientes pasos:

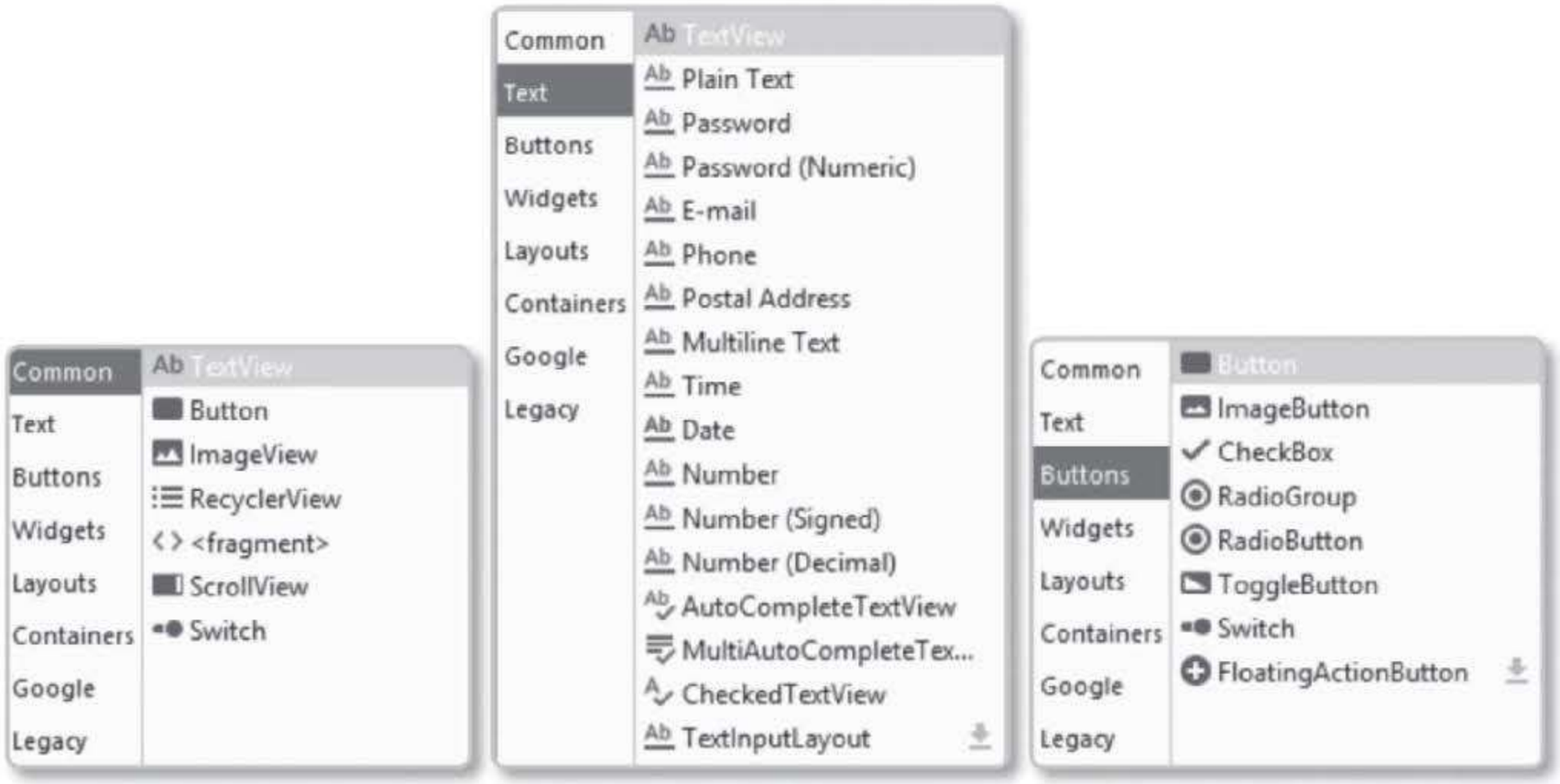
- Hacemos clic derecho sobre la clase.
- Seleccionamos **Delete**.



- Finalmente, presionamos el botón **OK**.

2.5 Declarar objetos de controles

El diseño de una aplicación Android incluye un conjunto de controles que permiten modelar un diseño de acuerdo con una necesidad. Por ejemplo, en un sistema de control de asistencia, se necesita ingresar las horas de trabajo o en un control de venta de productos que necesita para mostrar el total a pagar. Los objetos representan una instancia de la clase y, por intermedio de estas, se puede interactuar con las clases. Veamos una lista de algunos controles básicos para el diseño de una aplicación Android.



Cada uno de estos controles pertenece a una clase, el cual contiene las características necesarias para un proyecto de aplicación móvil. A continuación, se observa el formato para declarar objetos:

<Clase> **objeto1, objeto2, objeto3;**

donde, clase es la referencia de las características de los objetos. Un ejemplo se aprecia en la Tabla 8.

Tabla 8. Declaración de objetos

OBJETIVO	OBJETOS DECLARADOS
Definir los objetos para la clase EditText, la cual permita ingresar valores al proyecto.	EditText **etEmpleado, etHoras;**
Definir los objetos para la clase TextView, la cual permita mostrar mensajes estáticos al proyecto.	TextView **tvTitulo**;
Definir los objetos para la clase button, la cual permita mostrar botones de acción en el proyecto.	Button **btAceptar**, **btCancelar**;

2.6 Inicialización de variables

Cuando las variables son declaradas dentro de un método o son usadas como contadores o acumuladores, deben ser inicializadas con un valor según su tipo de datos. Si tomamos en cuenta que en Android todo se debe declarar; entonces, debemos aprender a inicializar todo tipo de valor. Debe considerar que el valor de inicialización debe ser del mismo tipo que se declara. El formato de inicialización es el siguiente:



donde, tipo de datos es la referencia del tipo de valor que se almacenará en la variable u objeto. Eso quiere decir que también se puede referenciar a una clase con el mismo formato. En la Tabla 9, se observa un ejemplo.

Tabla 9. Inicialización de valores y objetos

OBJETIVO	FORMAS DE INICIALIZACIÓN
Declarar la variable C de tipo entero e inicializarla con el mínimo valor positivo.	int c; c=0;
Declarar e inicializar la variable C de tipo entero con el mínimo valor positivo.	int c=0;
Inicializar el control etEmpleado al objeto declarado.	etEmpleado=(EditText) findViewById(R.id.*etEmpleado*); Debemos considerar que la función **findViewById** tiene por objetivo invocar al control colocado en el diseño de nuestra aplicación; esta referencia se hace mediante la referencia "R.id", el cual alcanza a todos los controles colocados en el diseño.

2.7 Conversiones

Cuando se efectúa dentro del código una asignación de variable, se debe tener en cuenta que el valor cumpla con el rango especificado por el tipo de datos. Esto se debe a una evaluación de compatibilidad que realiza el compilador. Existen dos tipos de conversiones:

- **Conversiones automáticas:** Esto se realiza cuando una variable de un determinado tipo se asigna a otro tipo con la restricción de que ambos tipos sean compatibles, siempre y cuando no exista pérdida de información o que el tipo de datos destino sea más grande que el origen. Veamos los siguientes casos:

```
//Aumentar dos puntos al promedio
int puntos = 2;
double promedio = 12.50;
promedio  = promedio + puntos;
```

la variable puntos se declara como tipo entero, mientras que promedio como tipo double; finalmente, promedio acumula el valor puntos en la variable promedio haciendo una conversión automática de la variable puntos a tipo double para que sean compatibles en la acumulación.

- **Conversión explícita:** Esto se realiza cuando un valor de tipo de datos de mayor capacidad es convertido a uno de inferior capacidad. A esto se le llama estrechamiento ¡, debido a que se estrecha explícitamente el valor para que quepa en el tipo de datos destino. Veamos el formato de conversión:

```
variable = (tipo) valor;
```

VALOR DE ASIGNACIÓN

TIPO DE DATOS A CONVERTIR

La variable tiene un tipo de datos específico, mientras que el valor tiene otro tipo no compatible. Asimismo, entre paréntesis se especifica el tipo de datos que sea compatible con el tipo de la variable para realizar una conversión correcta. Por ejemplo:

```
int n=100;
byte x=(byte)n;

la variable n de tipo entero tiene como valor 100 asignado y se necesita convertir a un tipo
byte, el cual es de menor capacidad. Es aquí donde se realiza la conversión, para lo cual se
emplea la técnica casting sobre el tipo de datos de la variable n.
```

Cuando se realizan conversiones, debemos conocer exactamente la capacidad de cada uno de ellos, ya que todos los tipos de datos se pueden convertir, pero ello no asegura una conversión segura entre estos. En la Tabla 10, se muestra la norma que se necesita para las conversiones sin pérdida de información.

Tabla 10. Norma de conversión

Tipo de datos origen	Tipo de datos destino
Byte	double, float, long, int, char, short
Short	double, float, long, int
Char	double, float, long, int
Int	double, float, long
long	double, float
float	double

2.8 Casting

Es considerado un procedimiento para transformar una variable primitiva de un tipo a otro, o para transformar un objeto de clase a otra clase siempre y cuando exista una relación de herencia entre ambos. Veamos el formato de aplicación de Casting a un control:

```
                    MÉTODO DE BÚSQUEDA DE CONTROL
                    ↓
objeto = (Clase)findViewById(R.Id.Control);
         ↑
         CASTING
```

donde, el metodo findViewById permite buscar los controles dentro del diseño de la aplicación y la cláusula R.Id especifica los controles obtenidos en el diseño. Veamos un ejemplo en la Tabla 11:

Tabla 11. Uso del método findViewById

OBJETIVO	FORMAS DE INICIALIZACION
Inicializa el control etEmpleado al objeto de la clase EditText.	**etEmpleado**=(EditText) findViewById(R.id.*etEmpleado*);
Inicializa el control tvImporte al objeto de la clase TextView.	**tvImporte**=(TextView) findViewById(R.id.*tvImporte*);
Inicializar el control btnCalcular al objeto de la clase Button.	**btnCalcular**=(Button) findViewById(R.id.*btnCalcular*);

2.9 Atributos

Es considerado como un elemento importante de la implementación de una clase. Estos elementos deben tener visibilidad privada por buenas prácticas de implementación y solo serán accesibles por medio de métodos públicos como get/set. Los atributos representan las características que puede tener una clase, y los valores contenidos en este objeto lo hacen diferente entre todos los objetos de la clase. Veamos la declaración de atributos dentro de la clase:

```
VISIBILIDAD
private <tipo> atributo;
TIPO DE DATOS o ESPECIFICACIÓN DE CLASE
```

donde, tipo es la referencia del tipo de valor que se almacenará en la variable u objeto. Eso quiere decir que también se puede referenciar a una clase con el mismo formato, por ejemplo, en la Tabla 12:

Tabla 12. Referencia a clases mediante Casting

OBJETIVO	FORMAS DE INICIALIZACION
Inicializa el control etEmpleado al objeto de la clase EditText.	**etEmpleado**=(EditText) findViewById(R.id.*etEmpleado*);
Inicializa el control tvImporte al objeto de la clase TextView.	**tvImporte**=(TextView) findViewById(R.id.*tvImporte*);
Inicializar el control btnCalcular al objeto de la clase Button.	**btnCalcular**=(Button) findViewById(R.id.*btnCalcular*);

2.10 Métodos get y set

Cuando se declaran atributos, hay una diferencia entre una declaración pública y otra privada, ya que el nivel de acceso se refleja al crear un objeto de la clase y tener acceso a los atributos. Entonces, podemos decir que el uso de los métodos get y set permitirán tener un mejor acceso a los atributos privados dentro de una clase.

2.10.1 Método set

Permite asignar un valor a un determinado atributo de clase. La idea principal consiste en dar un valor a un atributo que no es accesible por contar con visibilidad privada. Dentro de la clase, hace que un atributo privado tenga un valor desde el exterior. Su formato se presenta en la Figura 7:

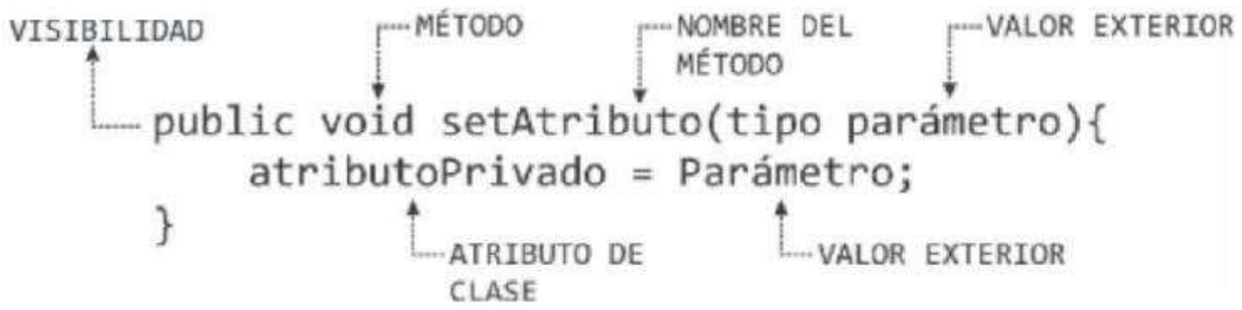


Figura 7. Implementación del método SET.

Donde:

- **Visibilidad:** Representa el alcance del método set; casi siempre se declara como public, ya que los valores provienen del exterior de la clase que lo implementa.
- **Método void:** La misión de este método es asignar un valor desde el exterior y no devolver nada desde el método.
- **setAtributo:** Es el nombre del método. Siempre se inicia con la palabra set, seguida del nombre del parámetro privado.
- **Valor exterior:** Es el valor recibido desde el exterior que será asignado al parámetro privado de la clase.
- **Atributo de clase:** Es la referencia al atributo privado definido en la clase.

Por ejemplo, si tenemos la declaración del atributo privado **Descripción** de la clase Producto y se necesita implementar su método set, podríamos tener las siguientes alternativas:

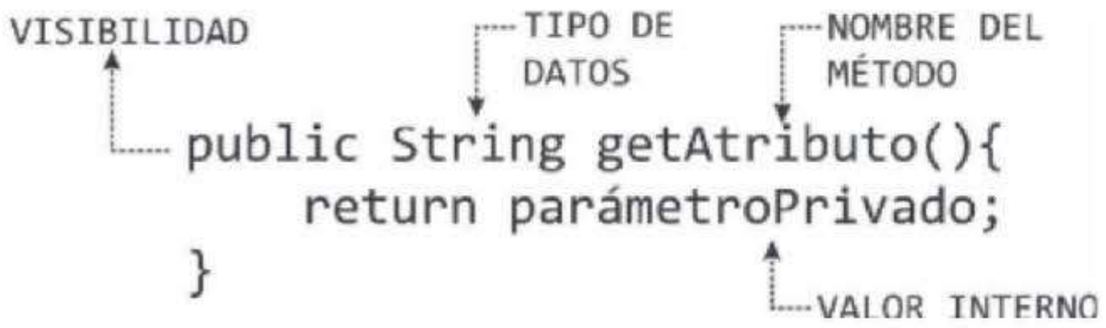
```
public class Empleado {
 private String descripcion;
}
La declaración del atributo descripción se realiza después de la creación
de la clase.

public String getDescripcion(){
return descripcion;
}
El método tiene como valor de devolución el atributo descripción.

public String getDescripcion(){
return this.descripcion;
}
La palabra reservada this hace referenciar a todos los atributos privados
de la clase y por este medio se puede especificar qué atributos se devolverán
por los métodos GET.
```

2.10.2 Método get

Si entendemos que los métodos set tienen la misión de asignar un valor a un atributo privado, es el método GET el que hace evidente la asignación devolviendo el valor asignado a dicho atributo



Donde:

- **Visibilidad:** Representa el alcance del método get. Siempre se declara como public, ya que este método será invocado desde fuera de la clase.
- **TipoDatos:** Es el tipo de datos del valor que devolverá el método. Hay que tener en cuenta que el parámetro privado y su método get deben tener el mismo tipo de datos.
- **getAtributo:** Es el nombre que se le asigna al método get. Casi siempre se inicia con la palabra get, seguida del nombre del parámetro privado.
- **Valor interno:** Aquí se especifica qué valor devolverá el método. Casi siempre se coloca el nombre del parámetro directamente; pero algunos programadores usan el operador this para hacer referencia a un miembro de la clase.

Por ejemplo, si tenemos la declaración del atributo privado **Descripción** de la clase **Producto** y se necesita implementar su método GET, podríamos contar con las siguientes alternativas:

```
public class Empleado {
 private String descripcion;
}
La declaración del atributo descripción se realiza después de la creación
de la clase.

public String getDescripcion(){
return descripcion;
}
El método tiene como valor de devolución el atributo descripción.

public String getDescripcion(){
return this.descripcion;
}
La palabra reservada this hace referenciar a todos los atributos privados
de la clase y por este medio se puede especificar qué atributos se devolverán
por los métodos GET.
```

2.10.3 Implementación de métodos get y set con Android Studio:

Se presenta la clase Producto con los atributos código, descripción, precio y stock, tal como se muestra en Figura 8.

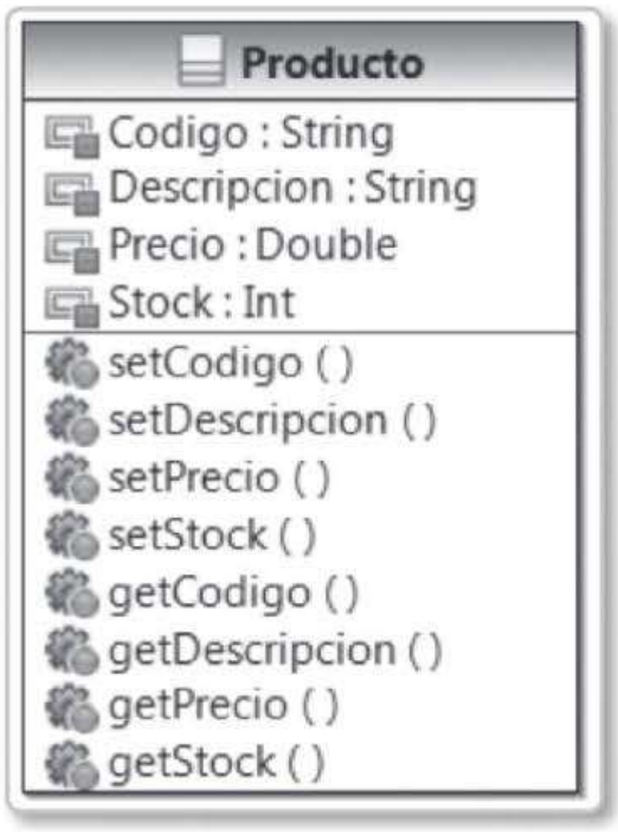


Figura 8. Ejemplo de Clase, con atributos y métodos.

Para poder crear los métodos get y set en Android Studio, debemos realizar los siguientes pasos:

1. Declaramos los atributos privados dentro de la clase producto.

```
package com.example.pjorientadaobjetos;

public class Producto {
    private String codigo;
    private String descripcion;
    private double precio;
    private int stock;

}
```

2. Presionamos clic derecho sobre la **clase > Generate** (ver Figura 9).

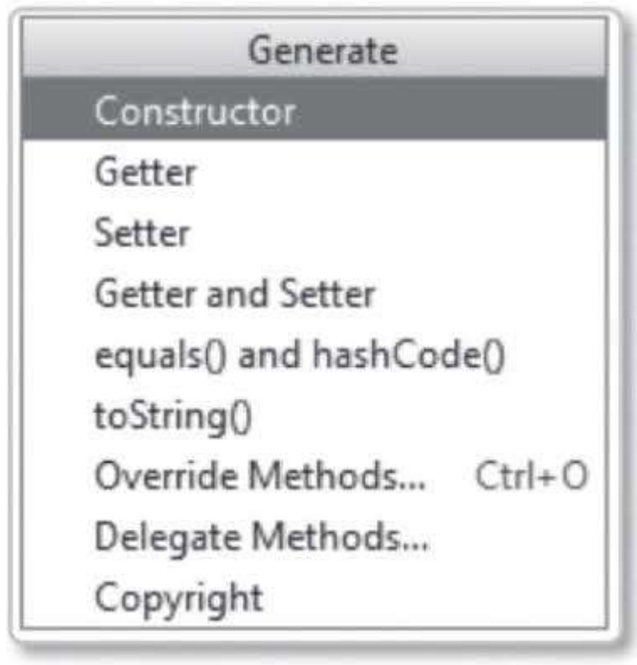


Figura 9. Opciones de Generate.

3. Seleccionamos **Getter and Setter** (ver Figura 10).

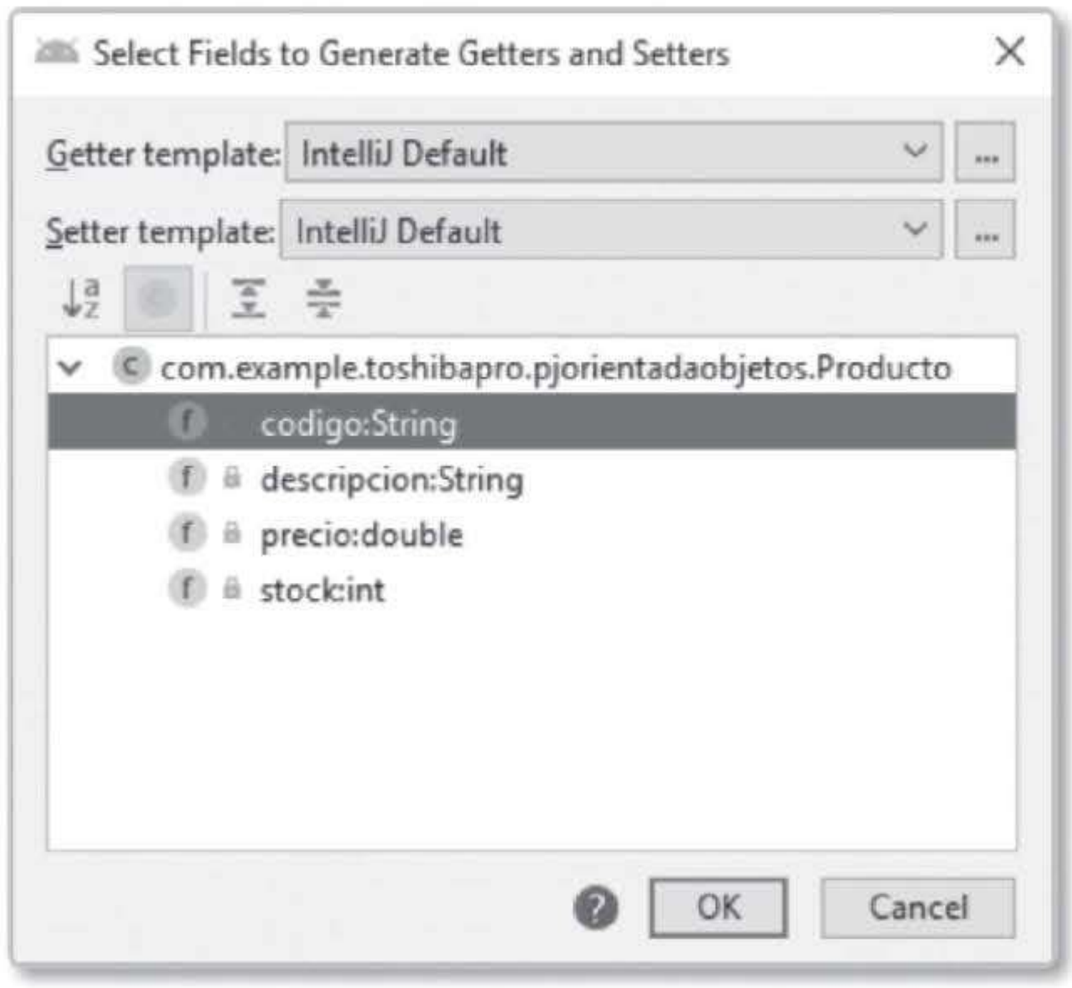


Figura 10. Opciones de Generate Getters and Setters.

4. Seleccionamos todos los atributos desde la lista ("Shift+Clic" en cada atributo) y presionamos OK. El código se muestra de la siguiente manera:

```
package com.example.pjorientadaobjetos;

public class Producto {
    private String codigo;
    private String descripcion;
    private double precio;
    private int stock;

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
```

```
        public int getStock() {
            return stock;
        }

        public void setStock(int stock) {
            this.stock - stock;
        }
    }
```

2.11 Métodos definidos por el usuario

Son métodos que permitirán realizar una determinada actividad según el criterio del programador. Hay que considerar que dichos métodos pueden ser usados en la clase donde se implementa o en las clases que se referencien.

2.11.1 Métodos sin valor de retorno

Son considerados como métodos vacíos y que no devuelven respuesta alguna a quien lo invoca, pero sí realizan una actividad dentro de la aplicación (ver Figura 11).



Figura 11. Definición de un método sin valor de retorno

Estos métodos se pueden implementar cuando se necesite realizar una asignación a un determinado parámetro o para actividades de impresión. A modo de ejemplo, veamos el método que permite asignar un valor a un parámetro dentro de la clase producto.

```
package com.example.pjorientadaobjetos;

public class Producto{
  //Atributo
  private String descripcion;

  //Metodo
  public void asignadescripcion(String des){
    descripcion - des;
  }
}
```

El método **asignaDescripcion** recibe como parámetro la descripción del producto y este será enviado al atributo privado 'descripción'. Desde la aplicación el código se observa de la siguiente manera:

```
Producto objPro - new Producto();
objPro.asignaDescripcion('Lavadora 16Kg');
```

2.11.2 Método con valor de retorno

Son métodos que devuelven información a quien los invoca. Solo debemos tener en cuenta que coincidan el tipo de salida del método y el tipo de datos del atributo que recepciona el valor. Cuenta con el siguiente formato (ver Figura 12):

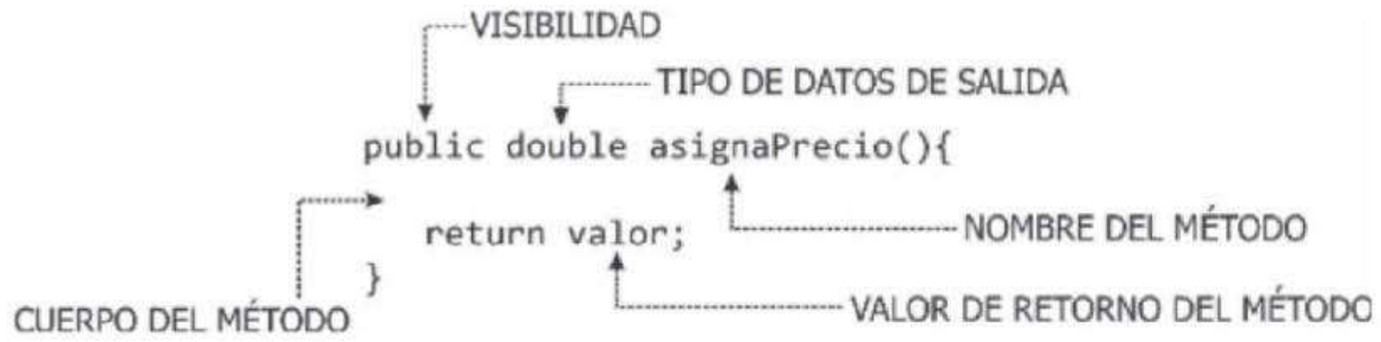


Figura 12. Definición de método con valor de retorno.

Podemos implementar estos métodos cuando se necesite realizar cálculos sobre determinados atributos de la clase. Veamos cómo asignar un precio a un determinado producto:

```
package com.example.pjorientadaobjetos;

public class Producto{
  //Metodo

  public double asignaPrecio(String descripcion){
    double precio=0;
    if (descripción.equals('Lavadora'))
        precio = 1500;
    return precio;
  }
}
```

El método **asignaPrecio** recibe como parámetro la descripción del producto para poder determinar de qué producto se trata y, usando la estructura selectiva If, se le asigna un monto tipo precio. Desde la aplicación, el código se observa de la siguiente manera:

```
Producto objPro = new Producto();
double precio = objPro.asignaPrecio('Lavadora');
```

2.10.3 Método constructor

Un constructor es un método especial dentro de la clase. Este se encargará de asignar valores iniciales a los parámetros privados de la clase. Hay que tener en cuenta que esta actividad solo lo realiza una vez por ejecución; vale decir que si necesita actualizar o modificar el valor asignado por el constructor, tendrá que usar los métodos set (ver Figura 13).

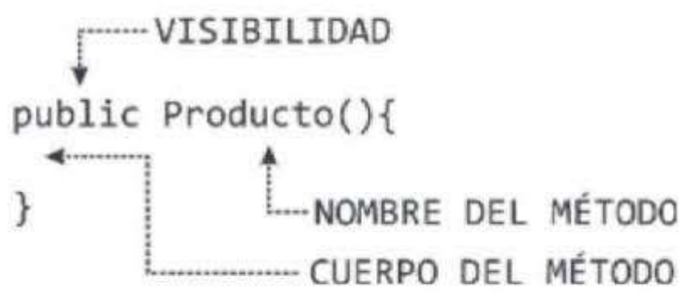


Figura 13. Definición de método constructor.

Donde:

- **Visibilidad:** Representa el alcance del método constructor. Casi siempre se declara como public, ya que este método será invocado desde fuera de la clase que se implementó.
- **Nombre del método:** Es el nombre asignado al método constructor. Para poder ser reconocido como constructor, debe tener el mismo nombre de la clase.
- **Cuerpo del método:** Aquí se pueden definir inicializaciones de atributos privados de clase.

Si contamos con el siguiente código...

```
package com.example.pjorientadaobjetos;

public class Producto{
      private String codigo;
      private String descripcion;
  private double precio;
      private int stock;
    }
```

... y se necesita implementar el método constructor de la clase Producto, podríamos usar el siguiente código:

```
public Producto(String codigo, String descripcion,
                double precio, int stock){
      this.codigo = codigo;
      this.descripcion = descripcion;
  this.precio = precio;
      this.stock = stock;
    }
```

CASO DESARROLLADO 01: **manejo de variables**

Implemente una aplicación móvil en Android que permita devolver el promedio de tres notas y que muestre un mensaje según el promedio con la siguiente escala (ver Tabla 13):

Tabla 13. Escala de valor según la condición

CONDICIÓN	MENSAJE
<=9.99	Desaprobado
Entre 10 y 12.99	Recuperacion
>=13	Aprobado

El prototipo de la aplicación se puede mostrar de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project...** desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 14):

Tabla 14. Especificación de valores para el proyecto

Create Android Project	Application Name	: pmCaso21
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_02
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar Activity Name Layout Name	: Empty Activity : MainActivity : activity_main

3. En el archivo **activity_main.xml**, realizamos el prototipo con el siguiente código:

activity_main.xml

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context=".MainActivity'>

    <TextView
        android:id='@+id/textView9'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='CONTROL DE PROMEDIO DE NOTAS'
        android:textColor='@color/colorNegro' />

    <TextView
        android:id='@+id/textView10'
        android:layout_width='match_parent'
```

```
        android:layout_height="wrap_content"
        android:text="NOTA 1"
        android:textColor="@color/colorNegro" />

    <EditText
        android:id="@+id/etNota1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="number" />

    <TextView
        android:id="@+id/textView11"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="NOTA 2"
        android:textColor="@color/colorNegro" />

    <EditText
        android:id="@+id/etNota2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="number" />

    <TextView
        android:id="@+id/textView12"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="NOTA 3"
        android:textColor="@color/colorNegro" />

    <EditText
        android:id="@+id/etNota3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="number" />

    <Button
        android:id="@+id/btnProcesar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="PROCESAR" />

    <TextView
        android:id="@+id/textView13"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="PROMEDIO"
        android:textColor="@color/colorNegro" />

    <TextView
        android:id="@+id/tvPromedio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="0.00"
        android:textColor="@color/colorNegro" />

    <TextView
        android:id="@+id/textView15"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="CONDICION"
        android:textColor="@color/colorNegro" />

    <TextView
        android:id="@+id/tvCondicion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="0.00"
        android:textColor="@color/colorNegro" />

</LinearLayout>
```

4. En el archivo **activity_main.java**, realizamos el proceso de la aplicación con el siguiente código:

MainActivity.java

```
package com.example.toshibapro.pjvariable;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    //Definicion de objetos
    EditText etNota1,etNota2,etNota3;
    Button btnProcesar;
    TextView tvPromedio,tvCondicion;

    //Declaracion de variables
    int nota1,nota2,nota3,nota4;
    String condicion;
    double promedio;

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

        //Obtener los valores del Layout
        etNota1 = (EditText) findViewById(R.id.etNota1);
        etNota2 = (EditText) findViewById(R.id.etNota2);
        etNota3 = (EditText) findViewById(R.id.etNota3);

        tvPromedio = (TextView) findViewById(R.id.tvPromedio);
        tvCondicion = (TextView) findViewById(R.id.tvCondicion);

        btnProcesar = (Button) findViewById(R.id.btnProcesar);
        btnProcesar.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                //Obtener los valores de los controles
                nota1 = Integer.parseInt(etNota1.getText().toString());
                nota2 = Integer.parseInt(etNota2.getText().toString());
```

```
                nota3 = Integer.parseInt(etNota3.getText().toString());

                //Calcular el promedio de notas
                promedio=(nota1+nota2+nota3)/3.0;

                //Determinar la condicion del alumno
                if (promedio<=9.99)
                    condicion='Desaprobado';
                else if (promedio<=12.99)
                    condicion='Recuperacion';
                else if(promedio<=20)
                    condicion='Aprobado';

                //Imprimir los resultados
                tvPromedio.setText(''+String.format('%.2f',promedio));
                tvCondicion.setText(''+condicion);
            }
        });
    }
}
```

5. En el archivo **colors.xml**, especificamos los colores con el siguiente código:

colors.xml

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
    <color name='colorPrimary'>#008577</color>
    <color name='colorPrimaryDark'>#00574B</color>
    <color name='colorAccent'>#D81B60</color>
    <color name='colorNegro'>#000000</color>
</resources>
```

CASO DESARROLLADO 02: programación orientada a objetos

Implemente una aplicación móvil en Android que permita devolver el importe a cobrar por un empleado, para lo cual debe ingresar su nombres, horas trabajadas y el costo por hora. Se debe crear una clase llamada Pago, que permite definir el proceso de cálculo de importe así como los métodos get y set de los atributos de la aplicación. El prototipo debe tener el siguiente aspecto:



Solución

1. Seleccionamos **File>New>New Project...** desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 15):

Tabla 15. Especificación de valores para el proyecto

Create Android Project	Application Name	: pmCaso22
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_02
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar	: Empty Activity
	Activity Name	: MainActivity
	Layout Name	: activity_main

3. En el archivo **activity_main.xml**, realizamos el prototipo con el siguiente código:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:gravity="center"
        android:layout_marginBottom="30dp"
        android:text="Control de Pago de Empleados"
        android:textColor="@android:color/background_dark" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="EMPLEADO"
        android:textColor="@android:color/background_dark" />

    <EditText
        android:id="@+id/etEmpleado"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:textColor="@android:color/background_dark" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="HORAS TRABAJADAS"
        android:textColor="@android:color/background_dark" />

    <EditText
        android:id="@+id/etHoras"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="number"
        android:textColor="@android:color/background_dark" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="COSTO POR HORA"
        android:textColor="@android:color/background_dark" />

    <EditText
        android:id="@+id/etCosto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberDecimal"
        android:textColor="@android:color/background_dark" />

    <Button
        android:id="@+id/btnCalcular"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="CALCULAR"
        android:textColor="@android:color/background_dark" />

    <TextView
        android:id="@+id/textView5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="IMPORTE A COBRAR"
        android:textColor="@android:color/background_dark" />

    <TextView
        android:id="@+id/tvImporte"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="0.00"
        android:textColor="@android:color/background_dark"
        android:textSize="24sp" />
</LinearLayout>
```

4. Agregamos una clase al proyecto llamado **Pago.java** (**File > New > Java Class**) y colocamos el siguiente código:

Pago.java

```
package com.example.toshibapro.pjorientadaobjetos;

public class Pago {
    private String empleado;
    private int horas;
    private double costo;

    public String getEmpleado() {
        return empleado;
    }
    public void setEmpleado(String empleado) {
        this.empleado = empleado;
    }
    public int getHoras() {
        return horas;
    }
    public void setHoras(int horas) {
        this.horas = horas;
    }
    public double getCosto() {
        return costo;
    }
    public void setCosto(double costo) {
        this.costo = costo;
    }

    public double calculaImporte(){
        return horas*costo;
    }
}
```

5. Modificamos el archivo **MainActivity.java** y colocamos el siguiente código:

MainActivity.java

```
package com.example.toshibapro.pjorientadaobjetos;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    EditText etEmpleado, etHoras, etCosto;
    Button btnCalcular;
    TextView tvImporte;

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

        //Obtener los valores del formulario
        etEmpleado = (EditText) findViewById(R.id.etEmpleado);
```

```
        etHoras= (EditText) findViewById(R.id.etHoras);
        etCosto = (EditText) findViewById(R.id.etCosto);
        tvImporte = (TextView) findViewById(R.id.tvImporte);

        btnCalcular = (Button) findViewById(R.id.btnCalcular);
        btnCalcular.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Pago objPago=new Pago();

                //Obtener los valores de los controles
                objPago.setEmpleado(etEmpleado.getText().toString());
                  objPago.setHoras(Integer.parseInt(etHoras.getText().
toString()));
                 objPago.setCosto(Double.parseDouble(etCosto.getText().
toString()));

                double importe=objPago.calculaImporte();

                tvImporte.setText(''+String.format('%.2f',importe));
            }
        });
    }
}
```

CAPÍTULO

3



Recursos y componentes Android

3.1 Estructura del proyecto Android

La creación de un proyecto Android conlleva un conjunto de elementos que permiten controlar la aplicación, pues existe una separación entre el código, el diseño y los recursos que utilizaremos para implementar una aplicación Android.

Los entornos de desarrollo tradicionales fuerzan al programador a crear un método de ordenamiento entre los elementos que se agregarán al proyecto. Android tiene todo ordenado; lo único que debemos saber es dónde se deben colocar ciertos elementos. Una vez que nos acostumbremos, entenderemos la idea de ordenamiento de Android. A continuación, en la captura y en la Tabla 16, veremos la estructura inicial de una aplicación Android.

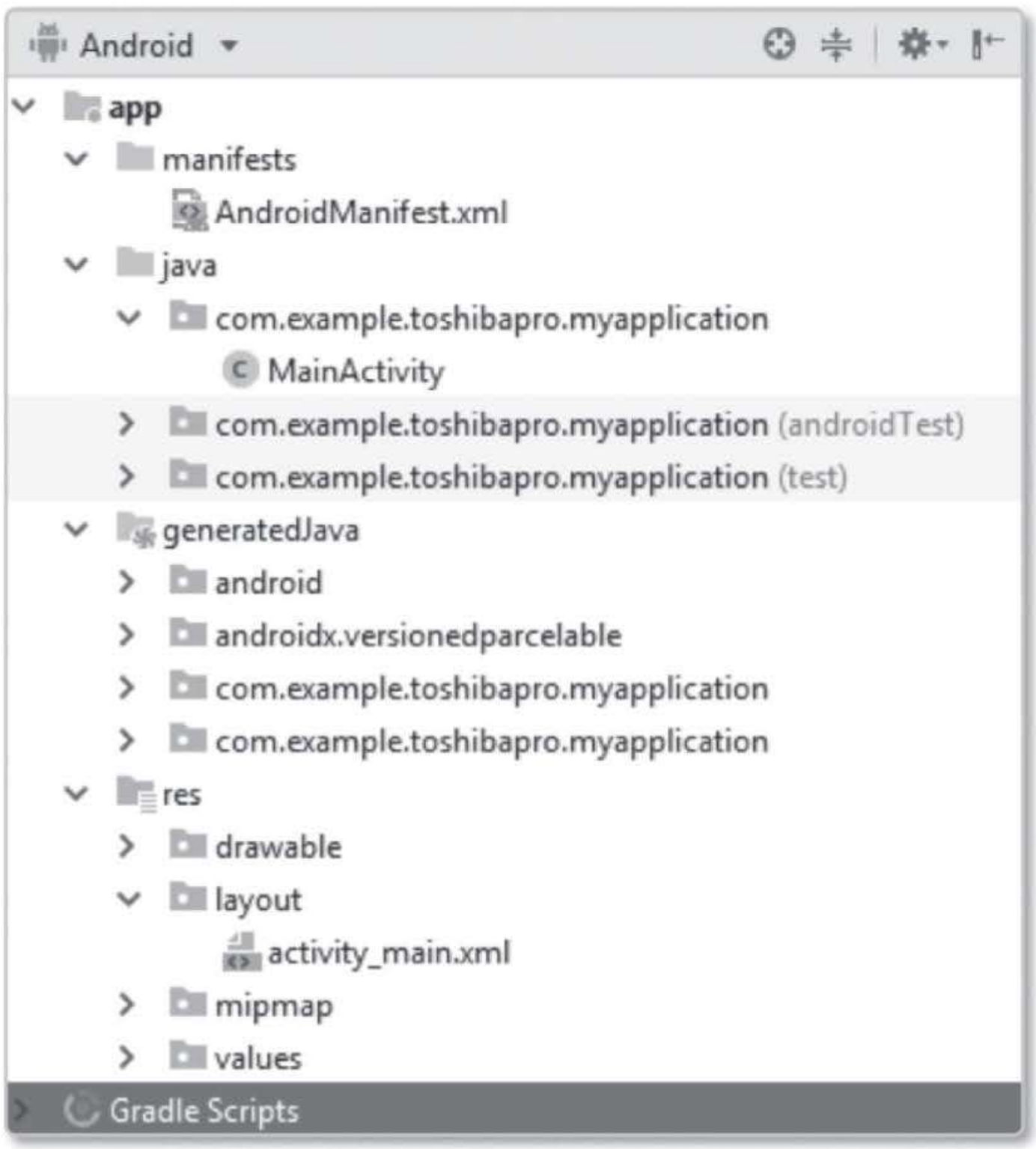


Tabla 16. Especificación de elementos Android

ELEMENTO	DESCRIPCIÓN
Manifest	Contiene el archivo **AndroidManifest**, el cual es considerado como un archivo de configuración de una aplicación Android. Debemos tener en cuenta que su estructura está compuesta por etiquetas XML.
Java	Contiene archivos con código Java que permiten aplicar dinamismo a una aplicación Android. Cada vez que agregue un **Activity Layout**, automáticamente se generara su entorno de desarrollo Java con el mismo nombre que la actividad.

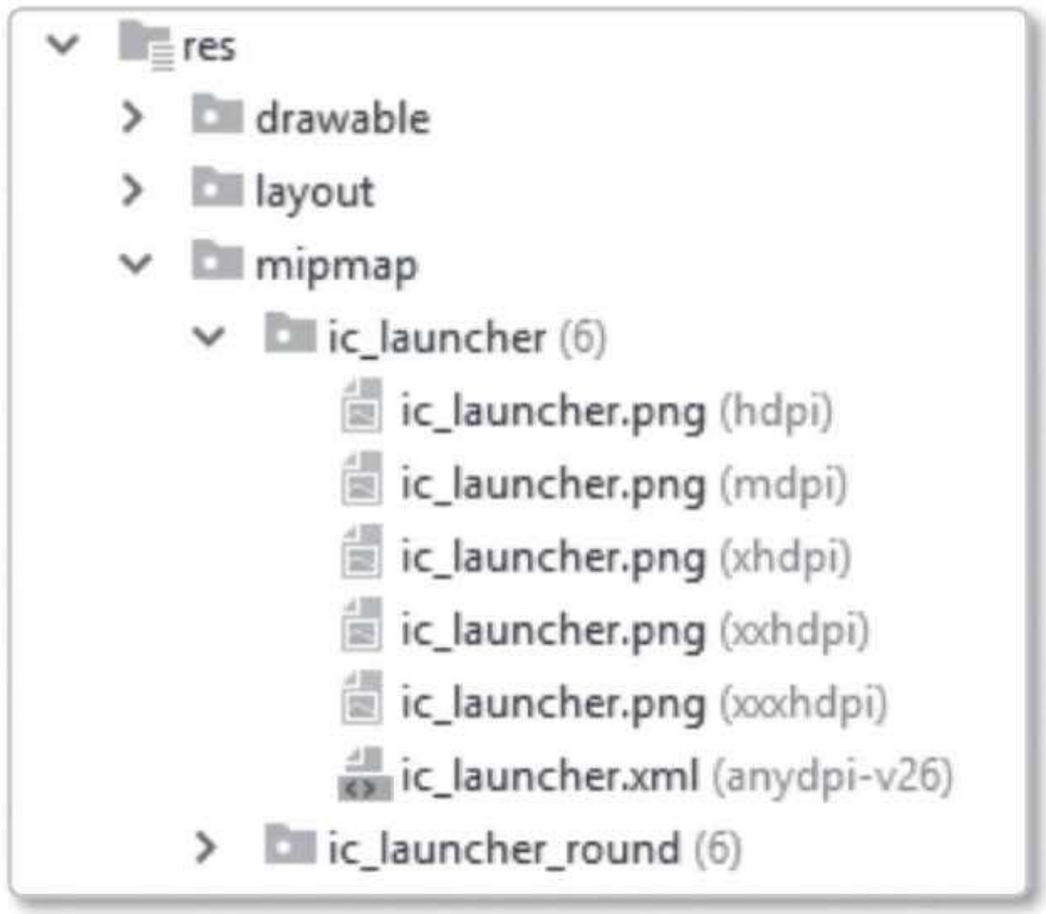
GeneratedJava	Presenta código autogenerado por la aplicación Android. Su principal archivo es **R.java**. Es un archivo que asocia todos los recursos que tenemos en nuestro proyecto al código Java.
Res	Es la agrupación de recursos que presenta una aplicación. Desde aquí podemos visualizar imágenes, pantalla de actividad y valores de una aplicación Android.
Gradle Scripts	Es considerado como un sistema de construcción que toma las mejores características de otros sistemas de compilación vistos en otros entornos de desarrollo y las combina en una sola. Se encuentra basado en la máquina virtual de Java, lo que significa que podemos escribir nuestra propia secuencia de comandos en Java.

3.2 Recursos

En este apartado se describe los principales recursos.

3.2.1 Mipmap

Contiene imágenes de tipo .png, las cuales actúan como íconos dentro de un proyecto Android. Estos se presentan en diversas resoluciones para que la aplicación se adapte a todos los dispositivos posibles.



Al presionar doble clic sobre una de las imágenes, se mostrará información del ícono:



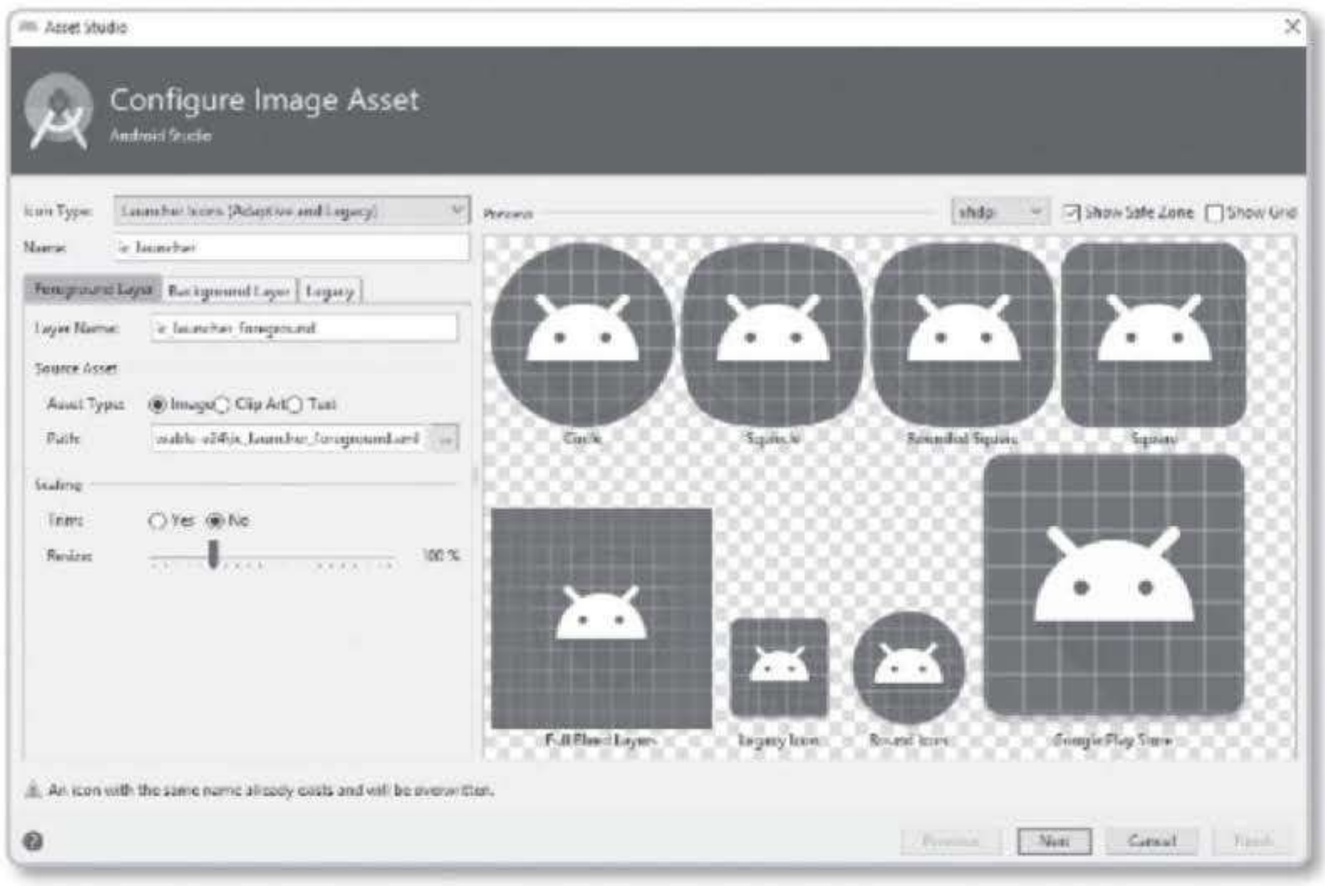
Agregaremos un ícono al proyecto con Android Studio. Para ello, se sigue los siguientes pasos:

1. Debemos tener una imagen de tipo .png para nuestro proyecto. En nuestro caso, hemos guardado la imagen en la unidad C.

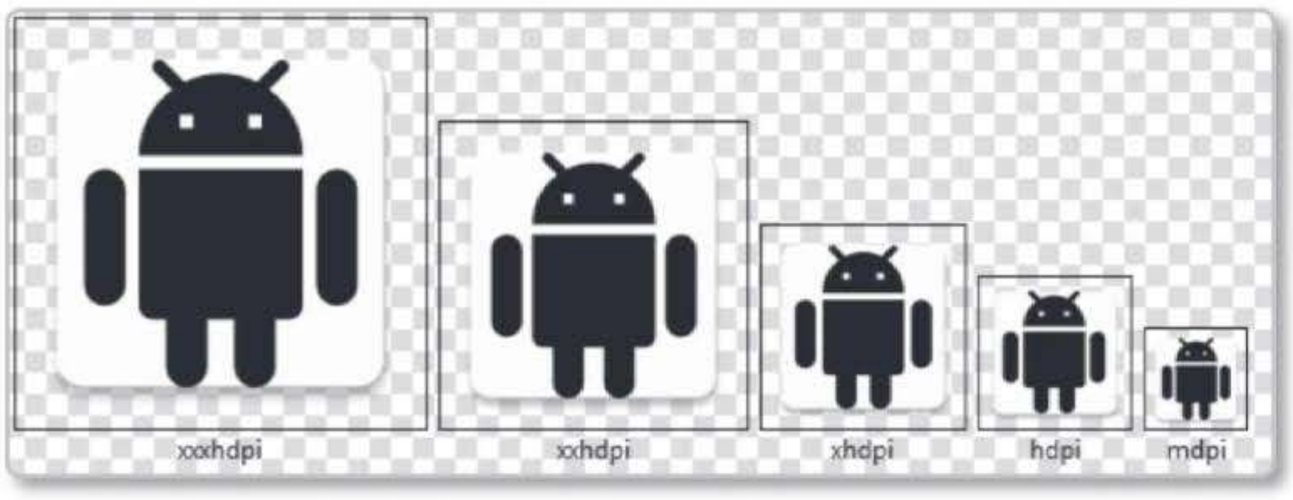


Restaurant.png

2. Hacemos clic derecho sobre la aplicación **app > New > Image Asset**.



3. Tenemos dos tipos de íconos: Launcher Icons(Adaptative), el cual presenta íconos adaptativos tanto dentro de la aplicación como íconos de acceso en su dispositivo móvil, y el Launcher Icons(Only Legacy), que presenta íconos de diferentes tamaños en un solo estilo, tal como se muestra en la siguiente captura:



4. Seleccionamos una imagen para su ícono desde la opción **Path** presionando clic sobre



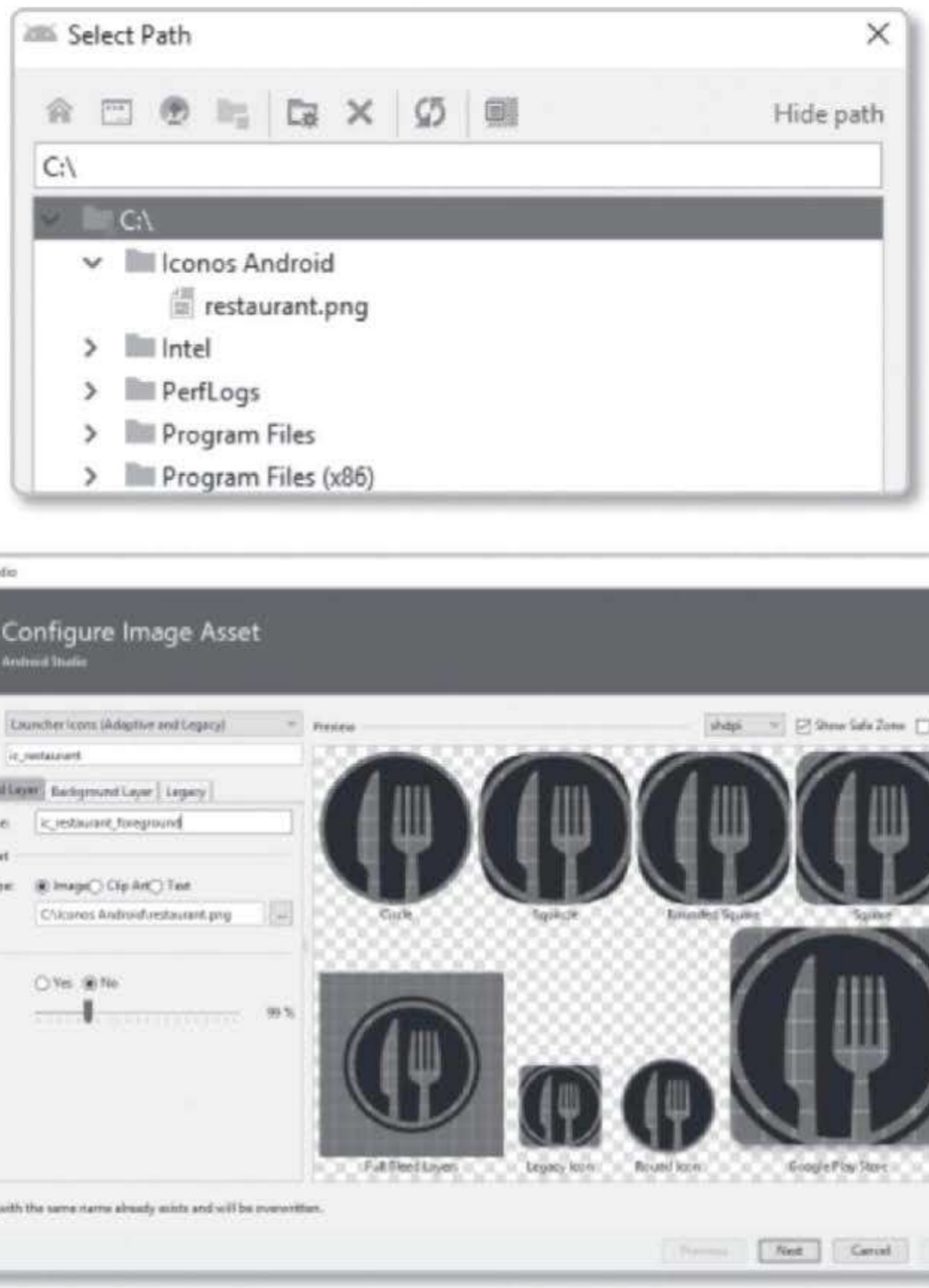
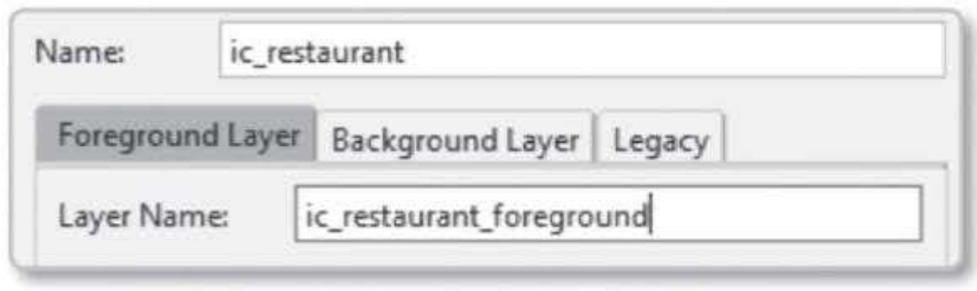


Fig. Imagen seleccionada como icono Android Studio

5. Una vez elegida la imagen, asignamos un nombre, tal como se muestra en la siguiente captura:



6. Finalmente, presionamos el botón **Finish** para crear los íconos en nuestro proyecto, tal como se muestra a continuación:

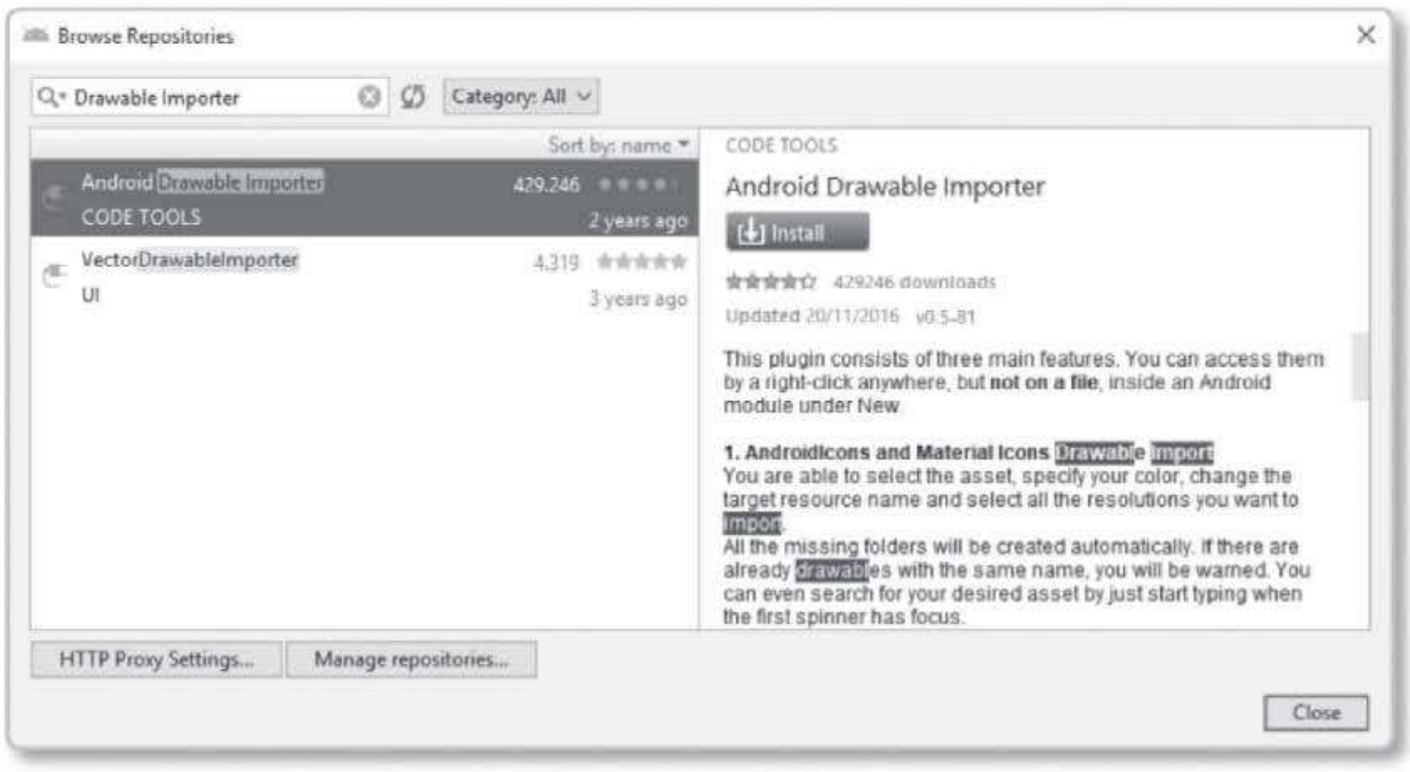


3.2.2 Drawables

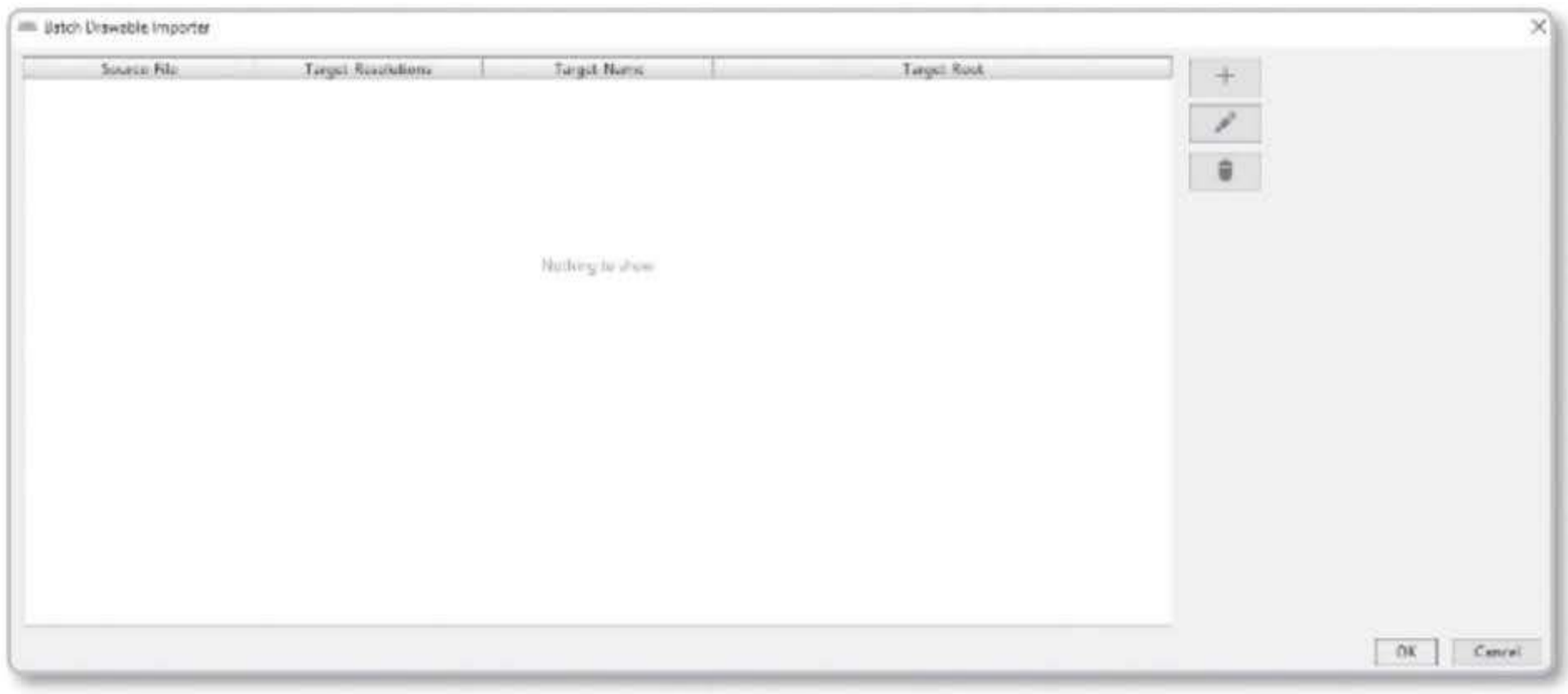
Aquí se almacenan todas las imágenes que necesite su aplicación. No debemos compararlo con la carpeta **mipmap**, pues las imágenes que usamos en nuestro proyecto no necesariamente usan un tamaño predeterminado como la hacen los íconos.

Para agregar un paquete de imágenes, como lo implementa Android para los íconos, debemos realizar los siguientes pasos:

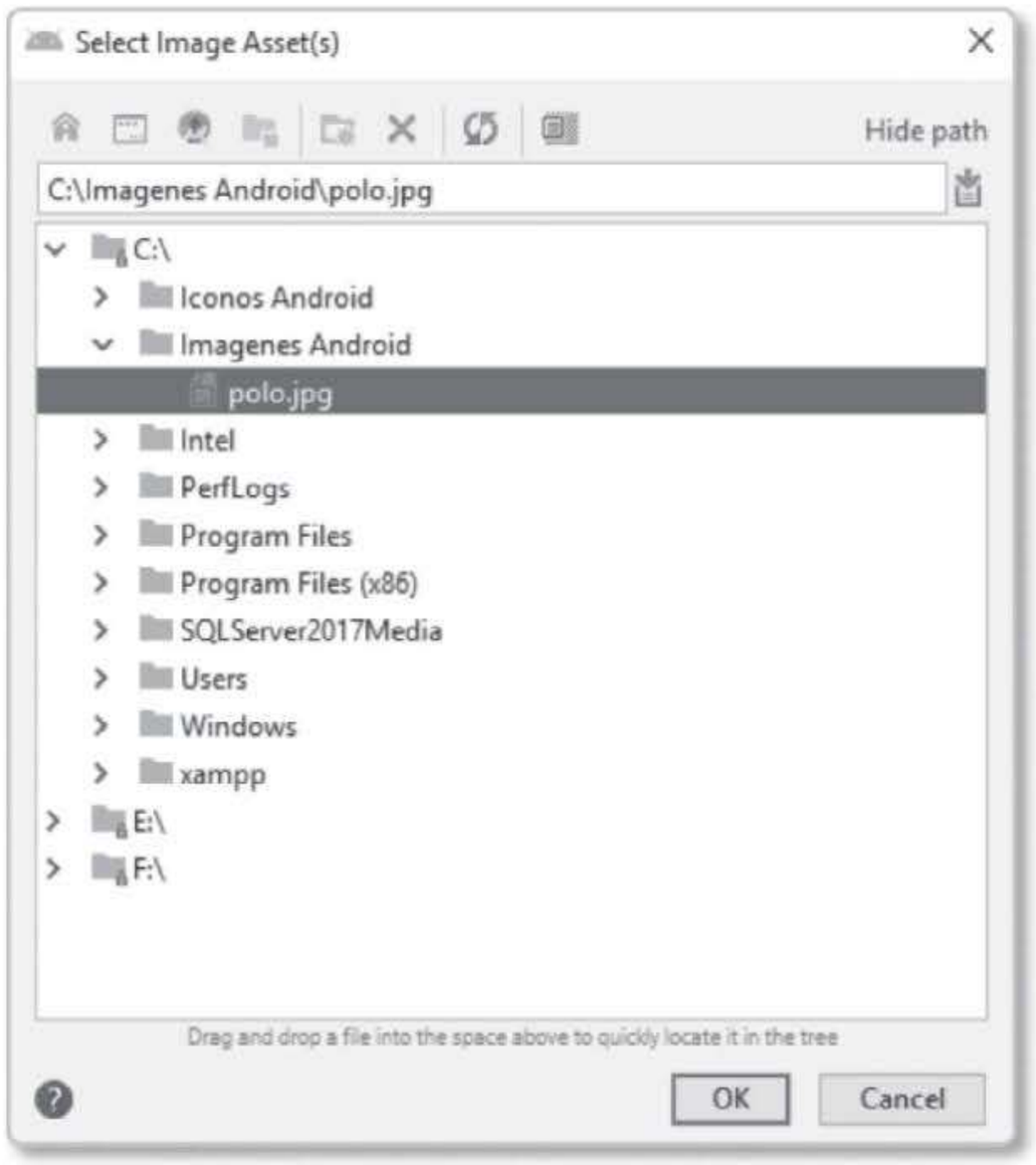
1. **File > Settings > Pluggins**
2. Colocamos, en la búsqueda, "Drawable Importer".
3. Presionamos clic sobre **Search in repositories**.



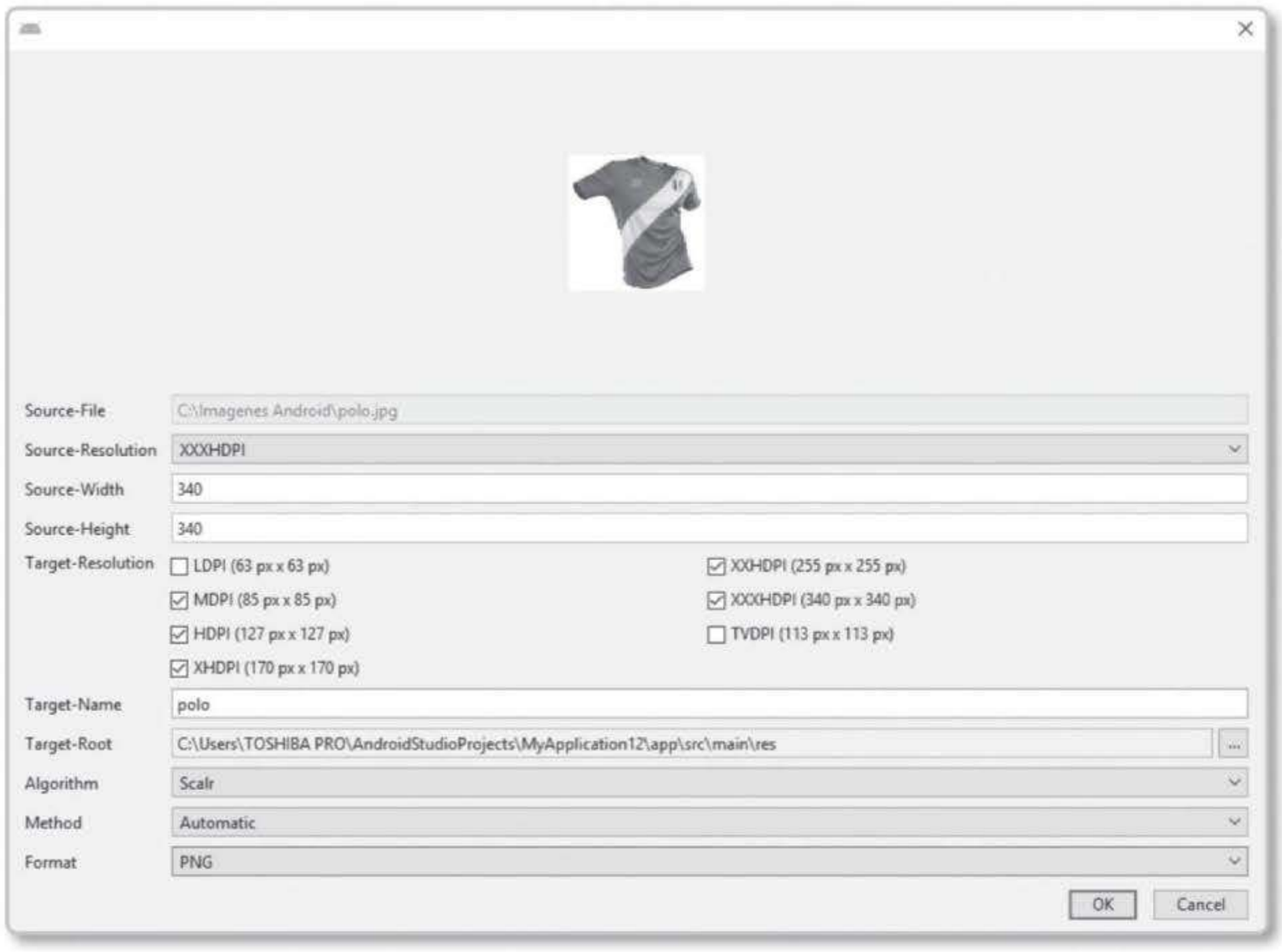
4. Seleccionamos el botón **Install**. Una vez instalado tendremos que cerrar el Android Studio y volver a ingresar.
5. Una vez reiniciado Android Studio, presionemos clic derecho sobre la aplicación **> New > Batch Drawable Import**.



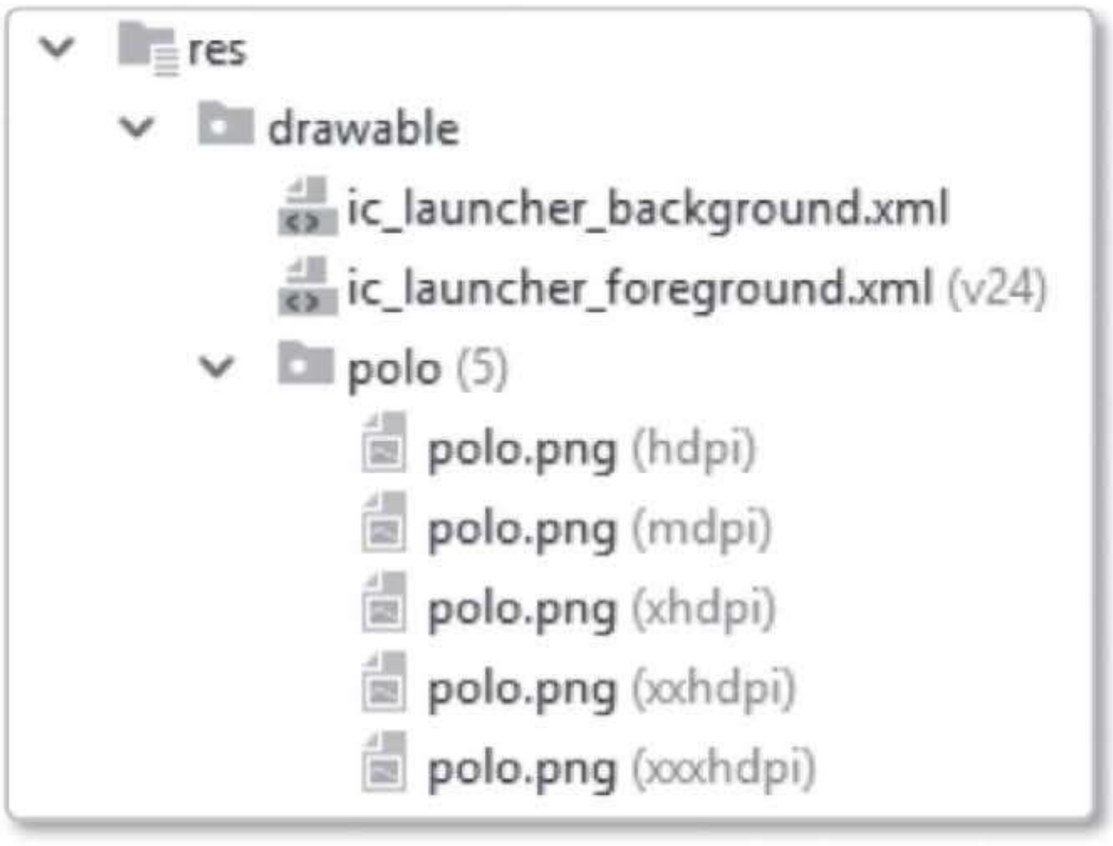
6. Para buscar la imagen, debemos presionar el botón **+**, y seleccionar la imagen como se muestra en la siguiente captura:



7. Luego, debemos configurar ciertos parámetros de la imagen, tal como se muestra a continuación.



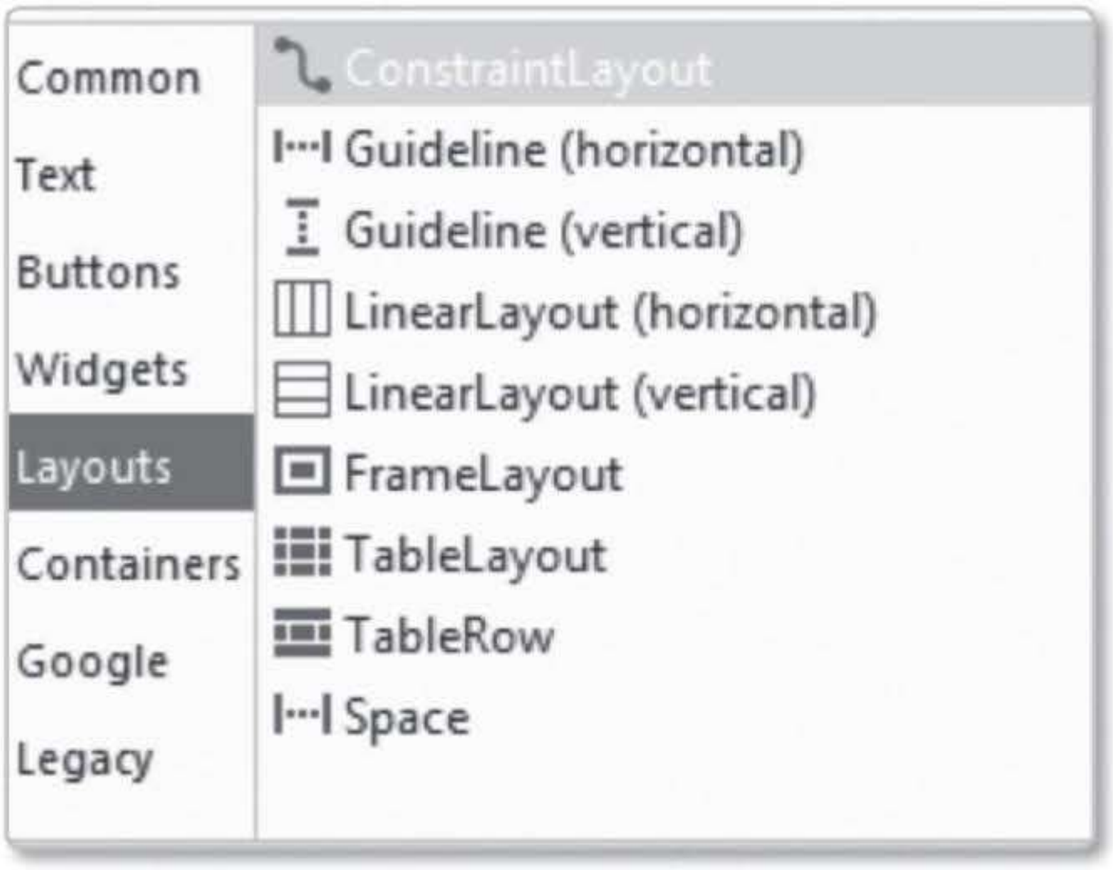
8. Finalmente, debemos comprobar que las imágenes se hayan creado dentro de la carpeta **Drawable**, tal como se muestra en la siguiente captura:



3.2.3 **Layouts**

Son elementos no visuales que sirven como base para diseñar una interfaz de usuario. Dentro de un layout, podremos colocar elementos visuales que permitirán diseñar un proceso de negocio para el usuario final.

En Android Studio, los tipos de layouts se muestran de la siguiente manera:



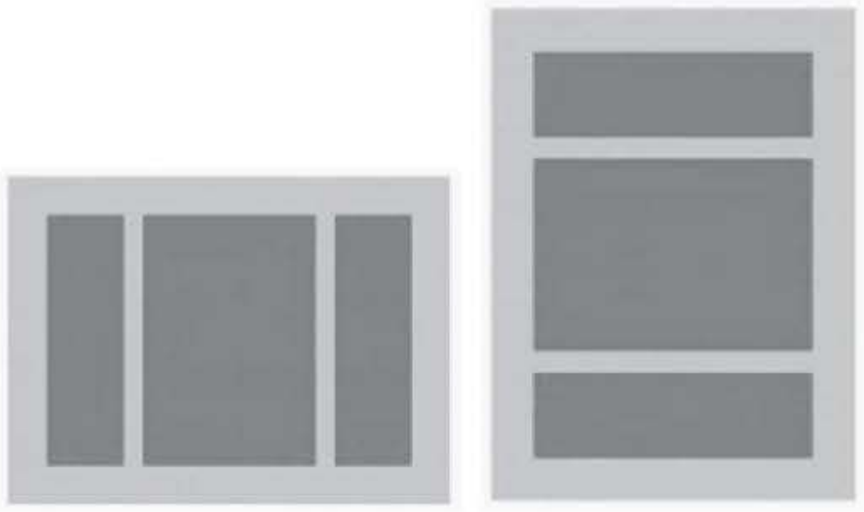
Asimismo, cuando se agrega un layout al entorno de diseño, se genera automáticamente un código que podrá ser administrable por el desarrollador. El código que se muestra pertenece a un LinearLayout de tipo vertical:

```
<LinearLayout
    android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="vertical"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp"
    tools:ignore="MissingConstraints" />
```

Como se aprecia, el código generado presenta ciertas propiedades con valores iniciales como el ancho y alto, orientación del layout, pero gráficamente no será visible:



LinearLayout. Posiciona los elementos en forma secuencial uno detrás de otro. El sentido puede variar tanto horizontal como verticalmente. Esto lo podrá realizar desde la propiedad **android: orientation**.



FrameLayout. Posiciona todos los elementos en la esquina superior izquierda, de tal forma que cada control quedará cubierto por el control siguiente. Se recomienda usar un FrameLayout para colocar un solo elemento. El siguiente código muestra la implementación del layout tipo FrameLayout.

```
<FrameLayout
    android:layout_width="368dp"
    android:layout_height="495dp"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">
</FrameLayout>
```

TableLayout. Posiciona todos los elementos de forma tabular. Para esto, define filas y columnas necesarias como si se tratara de una matriz.

```
<TableLayout
    android:layout_width="368dp"
    android:layout_height="495dp"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

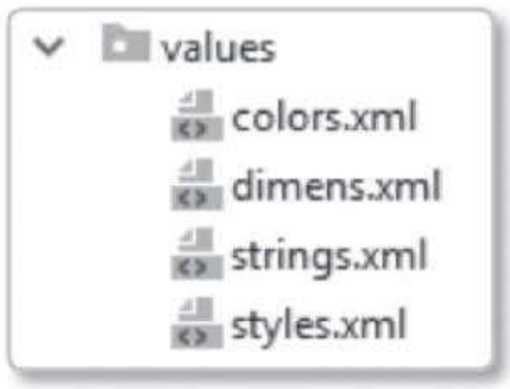
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</TableLayout>
```

Finalmente, se puede decir que, dentro de un mismo entorno, podemos combinar todos los tipos de *layout* dependiendo del nivel de diseño que necesite para su aplicación Android.

3.2.4 Values

Contiene los valores que serán necesarios dentro de nuestro proyecto Android. Esta declaración es parecida a una referencia global, en la cual todos los procesos de la aplicación podrán tener acceso a dichas variables. Estas podrán ser cadenas de texto, especificación de colores, especificación de dimensiones, estilos y arreglos. Android Studio presenta las siguiente categorías dentro de **Values**:



Colors

Aquí se definen, mediante un nombre, los colores que serán usados en todo el proyecto. Esto permite estandarizar el uso de los colores mediante una definición única. Debemos considerar que la definición de los colores se realiza por medio del código de color, los cuales podemos encontrar en el siguiente enlace: https://material.io/design/color/#tools-for-picking-colors

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
</resources>
```

Para definir un color, debemos agregar más etiquetas **<color>** dentro del archivo **colors.xml.** Definiremos los colores rojo, verde y azul.

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
    <color name='colorPrimary'>#008577</color>
    <color name='colorPrimaryDark'>#00574B</color>
    <color name='colorAccent'>#D81B60</color>
    <color name='rojo'>#FF0000</color>
    <color name='verde'>#00FF00</color>
    <color name='azul'>#0000FF</color>
</resources>
```

Para poder invocar a los colores, debemos seleccionar la propiedad que permite asignar color como **background, textColor**, etc.; luego, debemos colocar **@color** y seleccionar el color definido.

```
android:background='@color/rojo'
```

Por otro lado, también podríamos invocar al color desde el archivo java.

```
Resources objR = getResources();
int color = objR.getColor(R.color.rojo);
```

Strings

Permite definir cadenas que pueden usarse en cualquier parte de una aplicación Android.

```
<resources>
    <string name='titulo'>Manejo de Colores</string>
</resources>
```

Para poder invocar a las definiciones de cadena, debemos seleccionar la propiedad que permite asignar algún nombre o título de los controles. Luego, es necesario colocar **@string** y seleccionar la cadena definida.

```
android:name='@string/titulo'
```

Por otro lado, también podríamos invocar al color desde el archivo java.

```
Resources objR = getResources();
string cadena = objR.getName(R.string.titulo);
```

Styles

Permite definir un estilo que puede ser aplicado a cualquier control de una aplicación Android.

```
<resources>
    <style name='temaTitulo'>
        <item name='android:layout_height'>wrap_content</item>
        <item name='android:layout_weight'>wrap_content</item>
        <item name='android:textColor'>@color/colorAccent</item>
        <item name='android:textStyle'>bold</item>
    </style>
</resources>
```

Para invocar a las definiciones de estilos, debemos seleccionar la propiedad que permite asignar un estilo a los controles. Luego, se debe colocar **@style** y seleccionar el estilo definido.

```
<TextView
    android:id='@+id/textView'
    android:layout_weight='1'
    android:text='TextView'
    style='@style/temaTitulo'
/>
```

dimens: Permite definir las dimensiones que puede tener cualquier tipo de control en una aplicación Android.

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
    <dimen name='tamano_letra'>12dp</dimen>
</resources>
```

Para poder invocar las dimensiones, debemos seleccionar la propiedad que permite asignar una medida, ya sea alto, ancho o tamaño. Luego, se debe colocar **@color** y seleccionar el color definido.

```
android:textSize='@dimen/tamano_letra'
```

Por otro lado, también podríamos invocar al tamaño de texto desde el archivo java.

```
Resources objR = getResources();
float fontSize = objR.getDimension(R.dimen.tamano_letra);
```

3.3 Ciclo de vida de un activity

En una aplicación Android, encontraremos muchos elementos que permiten desarrollar un proceso. En ese sentido, el usuario interactúa mediante ventanas que llamaremos actividades. Estas controlan el ciclo de vida de una aplicación, pues el usuario muchas veces cambia de actividad, pero normalmente no la cierra con la posibilidad de volver. De esto se encarga el sistema, para lo cual genera una pila de actividades.

Android presenta como principal característica la destrucción de un proceso, ya que puede determinar cuándo será necesario destruir una aplicación. Para esto, presenta una serie de criterios que

son manejados directamente por el sistema, por ejemplo, la memoria que aún tiene disponible o la colisión de procesos (ver Figura 14).

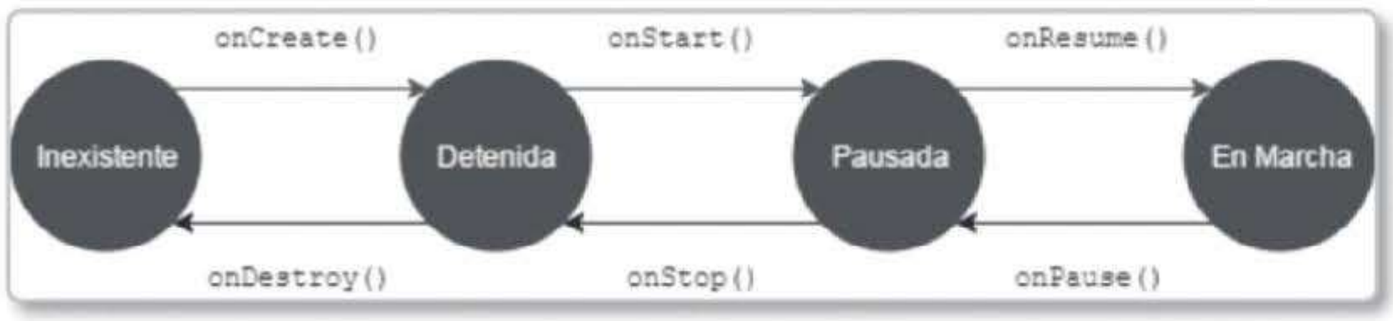


Figura 14. Ciclo de vida de una actividad. "Actividades en Android: concepto y ciclo de vida," por J. Revelo, 2015. Recuperado de http://www.hermosaprogramacion.com/2018/09/actividades-ciclo-de-vida/

3.3.1 Ciclo de vida

A continuación, en la Figura 15, se presenta el ciclo de vida ampliado de una actividad. Luego, se expone algunos elementos.

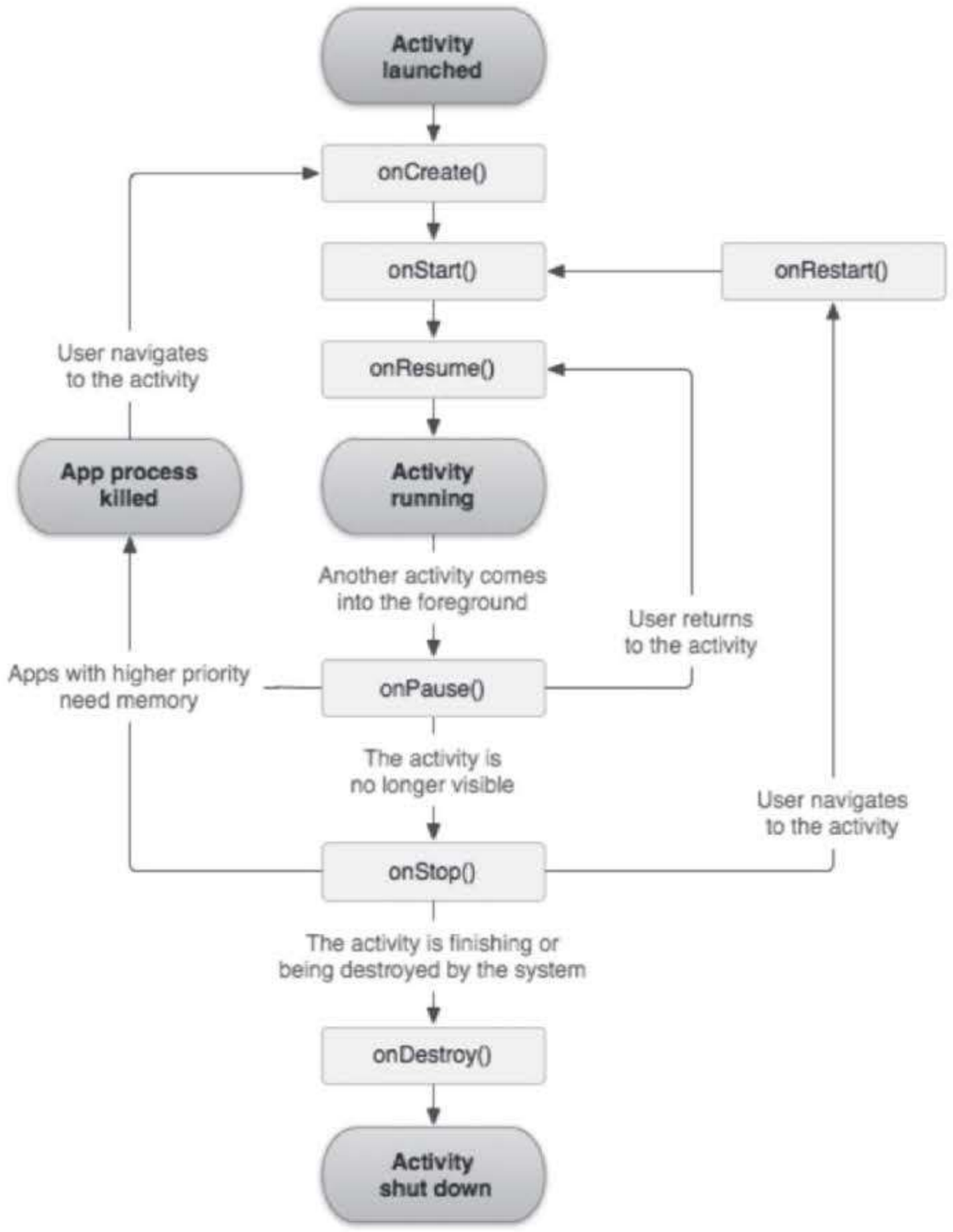


Figura 15. Ciclo de vida ampliado de una actividad. Tomado de "El ciclo de vida del Activity en Android," por E. Tijero, 2016. Recuperado de https://stories.devacademy.la/android-ciclo-de-vida-de-un-activity-b2a68c506961

- **onCreate:** Al crear una actividad, por primera vez es activado el método OnCreate(). Desde aquí se inicializan ciertos valores como la creación de vistas o el enlace de listas con datos. Normalmente, le sigue el método onStart().
- **onStart():** Es usado para indicar que la actividad está a punto de ser mostrada al usuario. Este método es llamado después del onCreate(). Se caracteriza por recibir una llamada justo antes de la visualización de una vista. Aquí le pueden seguir dos métodos: el OnResume, en caso de que la actividad pase a primer plano, o el OnStop(), si la actividad se oculta.
- **onResume():** Es usado antes de que la actividad se ponga "En Marcha" para interactuar con el usuario en primer plano. Normalmente, se emplea para ejecutar animaciones, iniciar la cámara, cargar datos actualizados, etc. Se caracteriza por recibir una llamada justo antes de que la actividad interactúe con el usuario, ya que esta actividad podría encontrarse en un pila de actividades. Normalmente, le sigue el método OnPause().
- **onPause():** Es empleado para indicar una pausa a la actividad actual con la posibilidad de que otra se reanude. Se pueden ejecutar acciones contrarias al método onResume(), como detener animaciones, música o almacenar los datos que estaban en edición, ya que despeja tareas que consumen el CPU del dispositivo. Aquí le pueden seguir dos métodos: el OnResume, en caso de que la actividad pase a primer plano, o el OnStop(), si la actividad se oculta.
- **onStop():** Es llamado cuando la actividad ya no va a ser visible para el usuario. Esto podría ocurrir por disminución de memoria o porque otra actividad se reanudó. Aquí le pueden seguir dos métodos: el OnRestart(), en caso de que la actividad vuelva a interactuar con el usuario, o el OnDestroy(), si la actividad desaparece.
- **onRestart():** Es llamado cuando la actividad vuelve a mostrarse al usuario por haber pasado por onStop(). Normalmente, le sigue el método OnStart().
- **onDestroy():** Se llama antes de que la actividad sea totalmente destruida. Este método es el último invocado y el que cierra el ciclo de vida de la actividad. Normalmente, se destruye una actividad cuando no hay memoria o si el usuario ha finalizado la aplicación invocando al método finish().

3.3.2 Estados

Android basa su desarrollo en el ciclo de vida; por eso, es necesario reconocer los estados por los que pueden pasar las actividades. Normalmente, son cuatro estados los reconocidos por Android:

- **Activa** (Running): Se considera activa cuando esta se encuentra encima de la pila de actividades; por lo tanto, lo hace visible al usuario.
- **Visible** (Paused): Se considera visible cuando aún se encuentra en la pila de actividades, pero no tiene el foco.
- **Parada** (Stopped): Se considera parada cuando la actividad no es visible para el usuario. Aquí se aprovecha en guardar el estado de la interfaz de usuario, preferencias o algunas modificaciones realizadas por el usuario.
- **Destruida** (Destroyed): Se considera destruida cuando la actividad finaliza por del método finish() o es destruida por el sistema.

Veamos el juego de eventos que se podrían generar por cambio de estado en las actividades (ver Tabla 17):

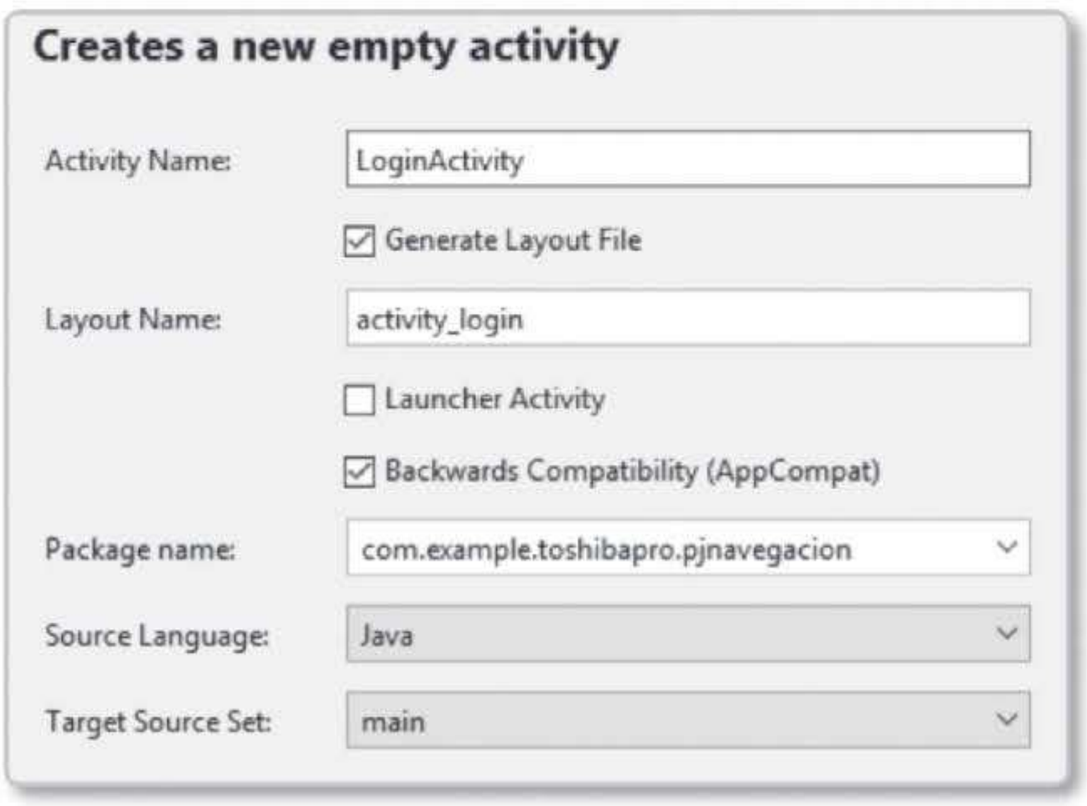
Tabla 17. Listado de eventos por cambio de estado en Android

EVENTO	JUEGO DE EVENTOS
Lanzar la Activity	Pasa de estado DESTRUIDO a estado ACTIVA, invocando a los siguiente métodos: onCreate() onStart() onResume()
Pulsar Back	Pasa de estado ACTIVA a estado DESTRUIDA, invocando a los siguientes métodos: onPause() onSaveInstanceState() onStop() onDestroy()
Pulsar Home	Pasa de estado ACTIVA a estado PARADA, invocando a los siguientes métodos: onPause() onSaveInstanceState() onStop()
Abrir la aplicación en recientes después de Home	Pasa de estado PARADA a estado ACTIVA, invocando a los siguientes métodos: onStart() onResume()
Abrir la aplicación en recientes después de Back	Pasa de estado DESTRUIDA a estado ACTIVA, invocando a los siguientes métodos: onCreate() onStart() onResume()
Rotar dispositivo	Pasa de estado ACTIVA a estado ACTIVA, invocando a los siguientes métodos: onPause() onSaveInstanceState() onStop() onDestroy() onCreate() onStart() onResume()
Lanzar otra Activity desde la Activity	Pasa de estado ACTIVA a estado PARADA, invocando a los siguientes métodos: onPause() onSaveInstanceState() onStop()
Pulsar Back en la otra Activity	Pasa de estado PARADA a estado ACTIVA, invocando a los siguientes métodos: onStart() onResume()

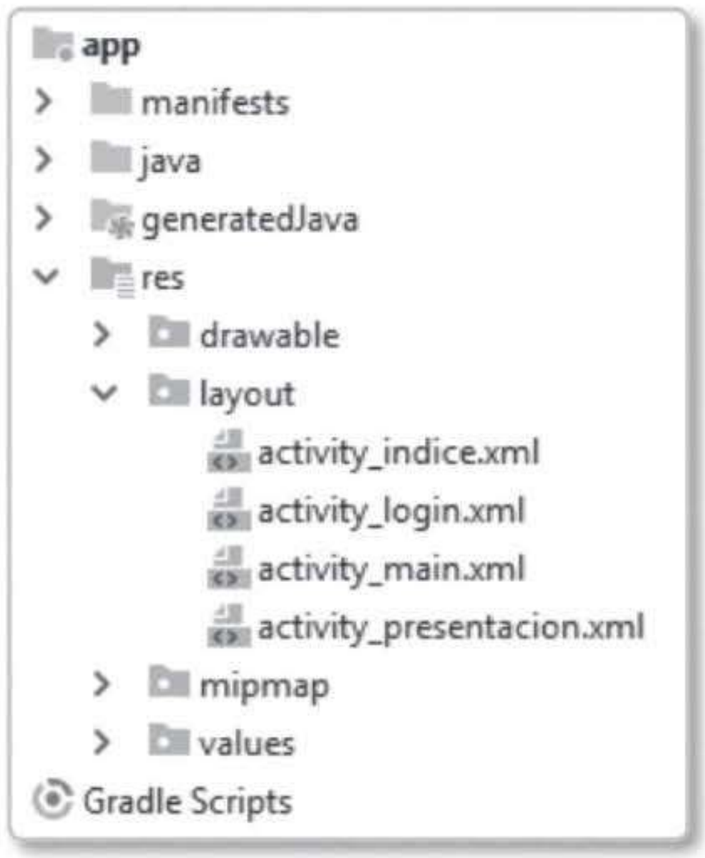
3.3.3 Navegación

Agregar una actividad. Abarca los siguientes pasos.

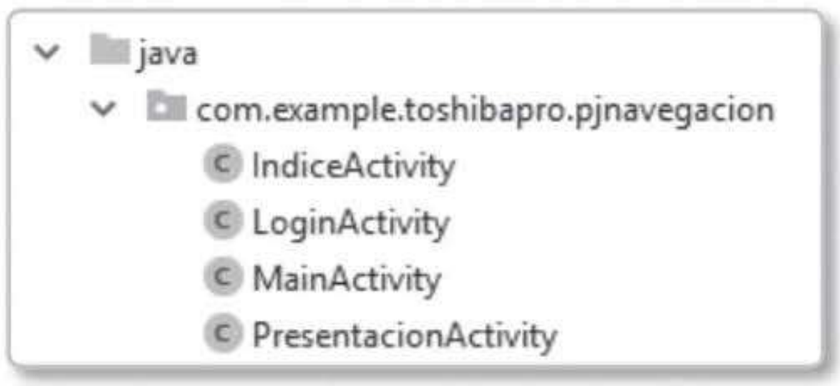
- Hacemos clic derecho sobre el proyecto **app > New > Activity > Empty Activity**
- Luego, debemos asignar un nombre a la actividad desde la opción **Activity Name**, tal como se muestra en la siguiente captura:



Debemos tener en cuenta que, al agregar actividades, estas se alojarán dentro de la carpeta **Layout** del proyecto, tal como se muestra a continuación.



Además, el hecho de agregar actividades genera archivos de tipo java para poder gestionar los controles de cada actividad, tal como se visualiza en la siguiente captura:



Registro de un activity en el archivo manifiest. Al agregar un activity automáticamente, se genera un código de implementación en el archivo **manifest** como un componente esencial del proyecto. El código se muestra de la siguiente forma:

Archivo: **AndroidManifest.xml**

```
<activity android:name=".MainActivity">
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

Como toda aplicación que contiene un conjunto de ventanas, se debe especificar quién debe iniciar el proceso. Para esto, se define un componente principal. En Android, el primer activity es el que se iniciará al ejecutar la aplicación. Para definir como principal, se debe agregar las etiquetas **<intent-filter>** y dentro especificar **android.intent.action.MAIN** en la etiqueta **<action>** y, finalmente, se debe especificar el mensaje de lanzamiento mediante la categoría **android.intent.category.LAUNCHER**.

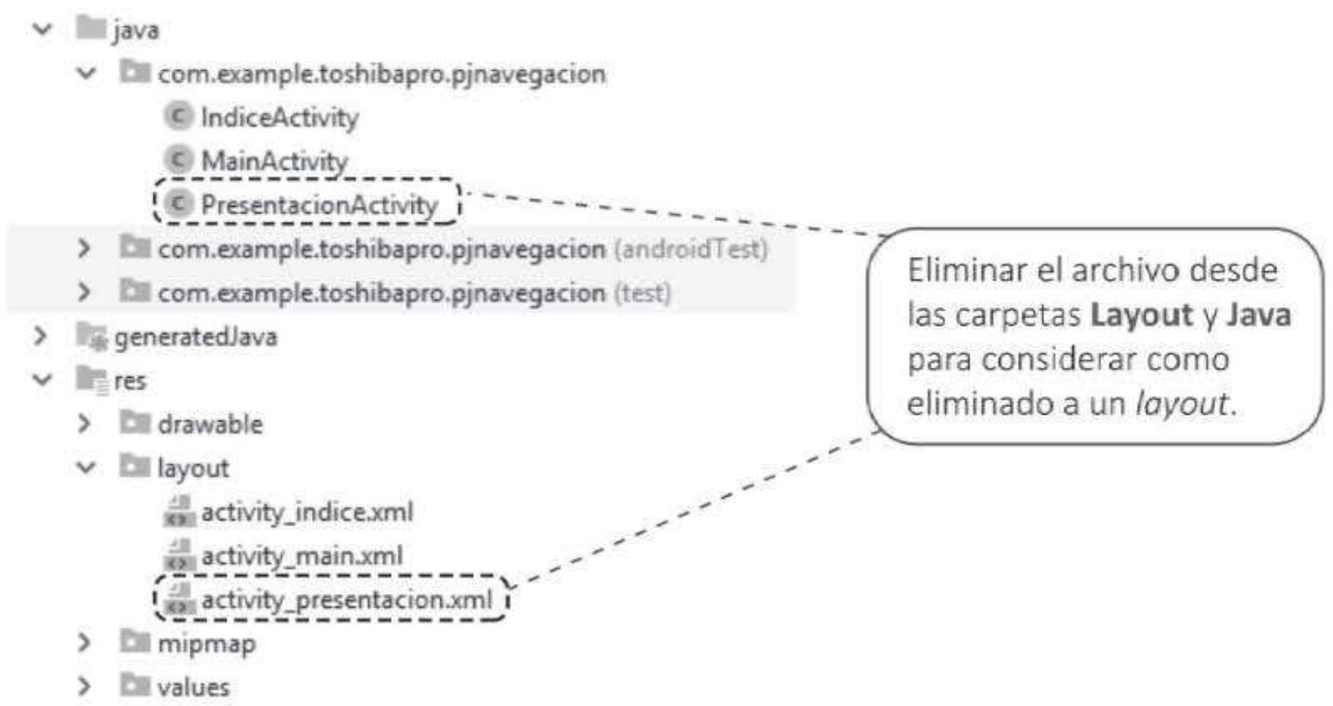
Cuando se agrega un *activity* al proyecto, el manifest solo muestra las etiquetas de creación, mas no la especificación de principal; pues, de todos los *activitys* que pueda agregar, solo uno podrá ser el principal. Veamos el código de los activitys agregados:

```
<activity android:name=".PresentacionActivity" />
<activity android:name=".IndiceActivity" />
```

Podríamos definir a la actividad **PresentacionActivity** como principal de la siguiente manera:

```
<activity android:name=".PresentacionActivity">
    <intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Eliminar una actividad. Debemos tener en cuenta que al eliminar una actividad también debemos eliminar el archivo .java generado por el agregado de un *activity*.



Seguimos los siguientes pasos para eliminar un *activity*.

- Hacemos clic derecho sobre la actividad que se encuentra dentro de la carpeta **layout**.
- Seleccionar **Delete**.
- Luego, debemos seguir los mismos pasos en la carpeta Java para eliminar la actividad con extensión java.

3.3.4 Uso del método onCreate de la actividad

El método onCreate define qué tareas debe realizar la actividad cuando esta se encuentre creada y activada. Inicialmente, el método onCreate se muestra de la siguiente manera:

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

Podemos hacer la prueba del método mostrando un mensaje por pantalla de la siguiente manera:

```
public class PresentacionActivity extends AppCompatActivity {

    Button btMensaje;

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

        //Instancia del boton
        btMensaje-findViewById(R.id.btMensaje);
        btMensaje.setOnClickListener(new View.OnClickListener() {
        @Override
       public void onClick(View v) {
        Toast.makeText(getApplicationContext(), 'Pantalla de Presentacion',
                       Toast.LENGTH_LONG).show();
        }
      });
   }
}
```

Se inicia declarando un objeto de la clase Button. Esto permitirá asociar el control agregado en el *activity*. Luego, se debe reconocer el control mediante el método findViewById. Finalmente, es necesario adicionar el evento setOnClickListener para agregar un mensaje mediante la notificación Toast.

3.4 Intents en la gestión de actividades

Dentro de una aplicación Android pueden ocurrir muchos eventos, como comunicarse entre actividades, enviar mensajes o activar múltiples opciones en las cuales el usuario puede seleccionar una alternativa. Todas estas actividades se pueden realizar gracias a los intents. Podríamos decir que

estos son considerados un mecanismo de comunicación tanto para una sola aplicación como para comunicar aplicaciones entre sí (ver Figura 16). Posteriormente, se indica los pasos involucrados.

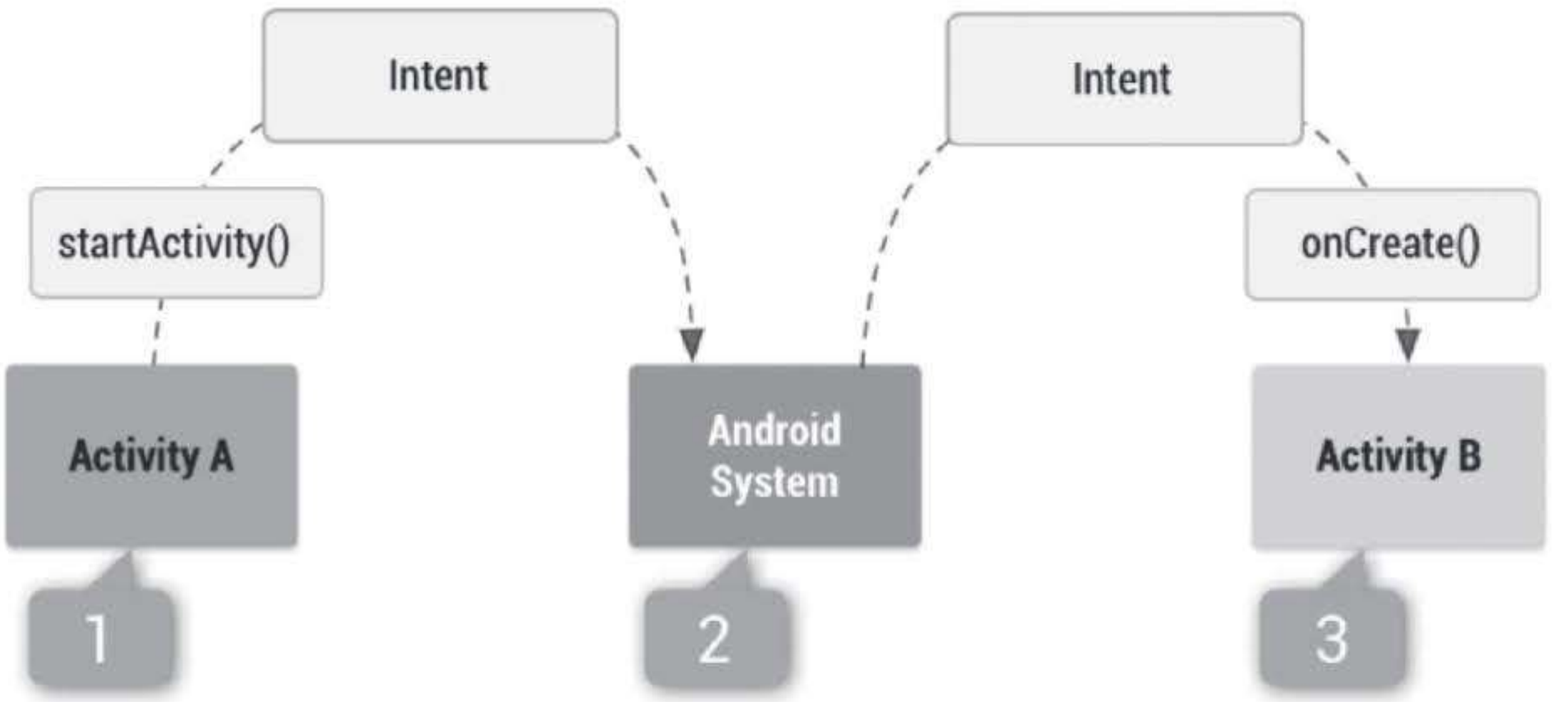


Figura 16. Mecanismo de comunicación de una aplicación Android. Tomado de "Intents y filtros de intents," por Developers, s.f. Recuperado de https://developer.android.com/guide/components/intents-filters?hl=es-419

1. Al iniciar la llamada a una actividad, esta se inicia y genera la intención.
2. El sistema operativo Android determina qué componente podría ejecutar para el intent.
3. Una vez detectado, se envía el proceso al método OnCreate y se inicia otra actividad.

Asimismo, se pueden presentar dos formas de intents:

Intents implícitos. Se les llama así cuando no se sabe exactamente qué componente se ejecutará, pero sí se sabe qué acciones podría tomar. Por ejemplo, en el dispositivo móvil, podríamos abrir un video y antes de que se muestre, nos aparecen las siguientes opciones (ver Figura 17):



Figura 17. Ejemplo de Intent Implícito. Tomado de "¿Cuál es el mejor reproductor de vídeo para Android? Analizamos a fondo los siete mejores," por I. Ramírez, 2017. Recuperado de https://www.xatakandroid.com/

La intención es abrir un video, pero debemos elegir entre los reproductores que se encuentran instalados en su dispositivo móvil. Por ejemplo, si queremos visualizar un tipo de

archivo y, al seleccionarlo, el usuario es direccionado a una página web, se presenta de la siguiente manera:

```
Intent - new Intent(Intent.ACTION_VIEW, Uri.parse('http://www.
editorialmacro.com'));
startActivity(intent);
```

Intents explícitos. Se les denomina así cuando se conoce exactamente qué componente se ejecutará. Una de sus tareas más importantes es la unión de actividades, ya que se tiene mucha afinidad con los controles dentro de la misma aplicación. Por ejemplo, si nos encontramos en la actividad **Menú Principal** y esta debe redireccionar mediante un botón hacia la actividad **Registrar Cliente**, el script debe mostrarse de la siguiente manera:

```
Intent intent - new Intent(MenuPrincipal.this,RegistrarCliente.class);
startActivity(intent);
```

Para el envío de parámetros entre actividades, se invoca al método putExtra. Por ejemplo, si la actividad principal envía información a la actividad **Registrar Cliente**, el script será de la siguiente manera:

```
Intent - new Intent(MenuPrincipal.this, RegistrarCliente.class);
intent.putExtra('usuario', usuario);
intent.putExtra('clave', clave);
startActivity(submitIntent
```

La especificación que colocamos entre comillas dobles son los valores que podrán ser recuperados desde la actividad destino. Por otro lado, las variables especificadas sin comillas son los valores que se están enviando como parámetros.

3.5 Manejo de Intent Filter

Si nuestra aplicación necesita que se inicie por otras aplicaciones, será necesario el uso de Intent-Filter, el cual será configurado en el AndroidManifest de la aplicación. A partir de la invocación del método startActivity es que el sistema Android determinará quién o quiénes pueden responder al *intent*.

Para definir adecuadamente correctamente los intent mediante filtros, cada filtro de *intent* debe ser lo más específico posible; por tanto, debemos tener clara la acción que debe realizar, la información que se necesita y la categoría de esta (ver Tabla 18).

Tabla 18. Action, Datos y Categoría de un Intent Filter

Acción	Se especifica qué acción se debe realizar. Estas pueden ser **ACTION_SEND o ACTION_VIEW**.
	
	`<activity android:name-'MainActivity'>`
	`  <intent-filter>`
	`  <action android:name-'android.intent.action.SEND'/>`
	`  </intent-filter>`
	`</activity>`

Datos	Se especifica una descripción de la información que está asociada con el intent de envío. Entre los valores que podemos especificar tenemos a MIME (extensión con propósitos múltiples de email) o URI (identificador de recursos).
	
	Por ejemplo, si especificamos en envío de información textual o de tipo imagen, podríamos colocar el siguiente script:
	
	`<activity android:name='MainActivity'>`
	`  <intent-filter>`
	`  <action android:name='android.intent.action.SEND'/>`
	`  <data android:mimeType='text/plain'/>`
	`  <data android:mimeType='image/*'/>`
	`  </intent-filter>`
	`</activity>`

Categoría	Se especifica un tipo adicional de propiedad para el intent. Normalmente, se especifica la opción predeterminada **CATEGORY_DEFAULT**. El script se podría mostrar de la siguiente manera:
	
	`<activity android:name='MainActivity'>`
	`  <intent-filter>`
	`  <action android:name='android.intent.action.SEND'/>`
	`  <category android:name='android.intent.category.DEFAULT'/>`
	`  <data android:mimeType='text/plain'/>`
	`  <data android:mimeType='image/*'/>`
	`  </intent-filter>`
	`</activity>`

3.6 Uso de PendingIntent

Especifica una acción que se puede ejecutar en el futuro. Es considerado como un token que le da a una aplicación externa el permiso para ejecutar un determinado código mientras se encuentra en una actividad. Entre las clases más usadas con la intención pendiente tenemos a NotificationManager, AlarmManager, Home Screen AppWidgetManager, entre otras aplicaciones. El siguiente script muestra cómo crear una instancia de la clase PendingIntent.

```
Intent - new Intent(this, AlarmaActivity.class);
PendingIntent pendingIntent - PendingIntent.getBroadcast(this, 0, intent, 0);
```

En nuestro ejemplo, se usa PendingIntent.getBroadcast() para llamar a un receptor de difusión para cuando la alarma suena, y dentro de ese receptor, se inicia el servicio para hacer sonar el dispositivo móvil. De otra manera, podríamos usar PendingIntent.getService() y llamar al servicio directamente cuando se active esa alarma.



3.7 Orientación de la pantalla

Toda aplicación que se muestra en un dispositivo móvil tiende a mostrarse de forma horizontal o vertical. Esto se produce por la naturaleza de la aplicación o por el movimiento del dispositivo. Todas estas actividades se realizan por un dispositivo interno llamado acelerómetro, que tiene la capacidad de detectar los movimientos del dispositivo (ver Figura 18).

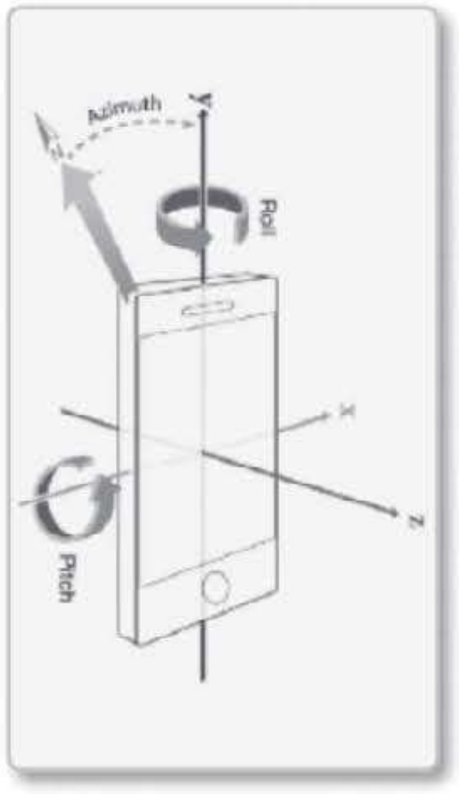


Figura 18. Modelo de acelerómetro. Tomado de "Qué es acelerómetro en un móvil y para qué sirve," por El Androide, s.f. Recuperado de https://elpoderdelandroide-verde.com/que-es-acelerometro-en-un-movil/

Veamos el script XML para asignar una orientación de tipo vertical(AndroidManifest.xml):

```
<activity android:name='.MainActivity' android:screenOrientation='portrait'>
```

Veamos el script XML para asignar una orientación de tipo horizontal(AndroidManifest.xml):

```
<activity android:name='.MainActivity' android:screenOrientation='landscape'>
```

Veamos el script Java para asignar una orientación de tipo vertical:

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

Veamos el script Java para asignar una orientación de tipo horizontal:

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

También, podríamos implementar una aplicación que inhabilite por defecto la orientación del dispositivo utilizando el siguiente script:

```
<activity android:name='.MainActivity'
android:configChanges='keyboard|orientation|keuboardHidden|screenSize' >
```

3.8 Tamaños de la pantalla

Toda aplicación ocupa un espacio dentro de la pantalla de nuestro dispositivo. Sin embargo, debemos considerar que la aplicación es la que proporciona un espacio que puede ser usado para distribuir los controles; es decir, nuestras aplicaciones deben adaptarse al dispositivo. Para esto, debemos usar un código que permita dicha adaptación (ver Figura 19).



Figura 19. Modelos de pantallas con Android. Tomado de "Soporte de diferentes pantallas," por Desarrollador Android, 2015. Recuperado de https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/soporte-de-diferentes-dispositivos/soporte-de-diferentes-pantallas/

Android se ejecuta en cualquier tipo de dispositivo y en diferentes tamaños o densidades de píxeles, lo cual se debe a que su sistema permite adaptar una interfaz de usuario a diferentes pantallas. No obstante, debemos configurar ciertos controles para su adaptación total, de modo que tenga una presentación correcta para el usuario.

Densidades de píxeles. Es la cantidad de píxeles que presenta un sector de la pantalla. Es más conocido como ppp (puntos por pulgada) o ppi (pixels per inch). La densidad de píxeles es un dato calculado en función de la resolución y el tamaño de la pantalla, que nos permite configurar de manera correcta la distribución de tamaños y así ganar la mejor resolución dentro de una aplicación (ver Figura 20).

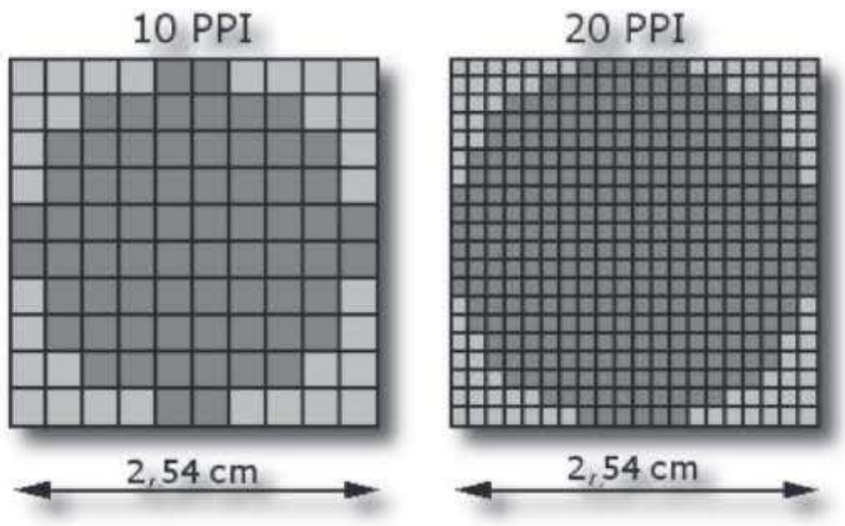
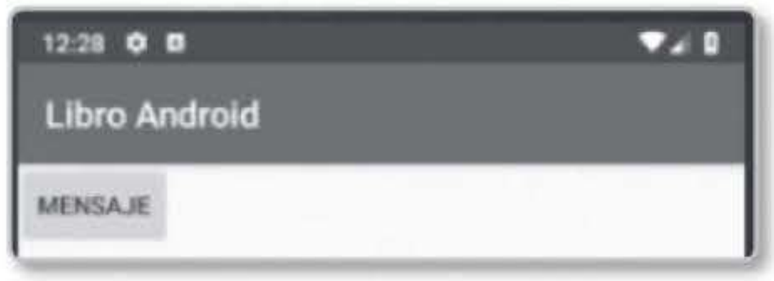


Figura 20. Densidad de pixeles. Tomado de "Qué es la densidad de píxeles y en qué te afecta," por The Phone House Spain, 2017. Recuperado de https://blog.phonehouse.es/2017/02/01/que-es-la-densidad-de-pixeles-y-en-que-te-afecta/

Adaptar tamaño con wrap_content. Asigna un determinado tamaño en ancho o alto dependiendo del valor que contenga. Por ejemplo, si colocamos un botón y asignamos el siguiente script...

```
<Button
    android:id='@+id/btMensaje'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:text='MENSAJE' />
```

... el botón adaptará el ancho de acuerdo con la palabra MENSAJE, y el botón quedará en el diseño de la siguiente forma:



Adaptar tamaño con match_parent o fill_parent. Asigna un determinado tamaño en ancho o alto dependiendo del tamaño de la vista que lo contenga. Por ejemplo, si colocamos un botón y asignamos el siguiente script...

```
<Button
    android:id='@+id/btMensaje'
    android:layout_width='match_parent'
    android:layout_height='wrap_content'
    android:text='MENSAJE' />
```

... el botón adaptará el ancho de acuerdo con el control **LinearLayout** que lo contiene, y el botón quedará en el diseño de la siguiente forma:



Unidades de medida. Para una mejor asignación de tamaños, debemos analizar las unidades de medida con las cuales Android asigna a sus controles. Veamos la siguiente lista (ver Tabla 19).

Tabla 19. DENSIDAD- ESPECIFICACIÓN

UNIDAD DE MEDIDA	DESCRIPCIÓN
dp	**Density Pixel** Es una unidad que se basa principalmente en la densidad física que puede tener la pantalla de su dispositivo móvil. Esto se debe a los diferentes tamaños que presentan los dispositivos móviles en que se ejecutarán las aplicaciones. Su aplicación se produce como si estuviéramos tratando a un pixel, con la diferencia de que, si su dispositivo presenta poca densidad, se usarán menos píxeles; sucede al revés si el dispositivo presenta una densidad alta. Es por esta razón que las aplicaciones móviles no se distorsionan cuando se visualizan en diferentes dispositivos.
sp	**Scale Pixel** Su forma de trabajo es similar a la unidad dp, pero adiciona una escala automática según el tamaño de la fuente que asigne en una aplicación móvil.

pt	**Points** Representa solo el 72% de una pulgada y la resolución dependerá mucho del tamaño físico que presente el dispositivo móvil
px	**Pixel** Representa a un píxel en una resolución real de la pantalla del dispositivo. Se recomienda no hacer uso de dicha medida en aplicaciones móviles, porque estos presentan una variedad de píxeles por pulgada.

3.9 Creación de fragmentos

Los fragmentos nacen por la necesidad de adaptar los contenidos de una aplicación en cualquier tipo de dispositivo. A partir de Android 3.0, se incorpora dicha necesidad por la amplia gama de dispositivos móviles y principalmente por la orientación de las actividades (horizontal y vertical).

Una actividad puede contener muchos fragmentos. Esto permite optimizar el diseño de una aplicación y hace que la distribución de los componentes tenga una nueva perspectiva frente al usuario final.

Podemos mencionar algunas de las ventajas que genera el uso de los fragmentos en una aplicación móvil:

- Permite la reutilización de su código.
- Permite ahorrar tiempo en la implementación del diseño.
- Facilita el despliegue de aplicaciones en cualquier tipo de dispositivo sin que importen el tamaño y la orientación.
- Posibilita el diseño de actividades dinámicas, porque permite gestionar mejor los componentes de navegación y otros elementos.

Ciclo de vida de un fragmento. En la Figura 21, se presenta el ciclo de vida de un fragmento.

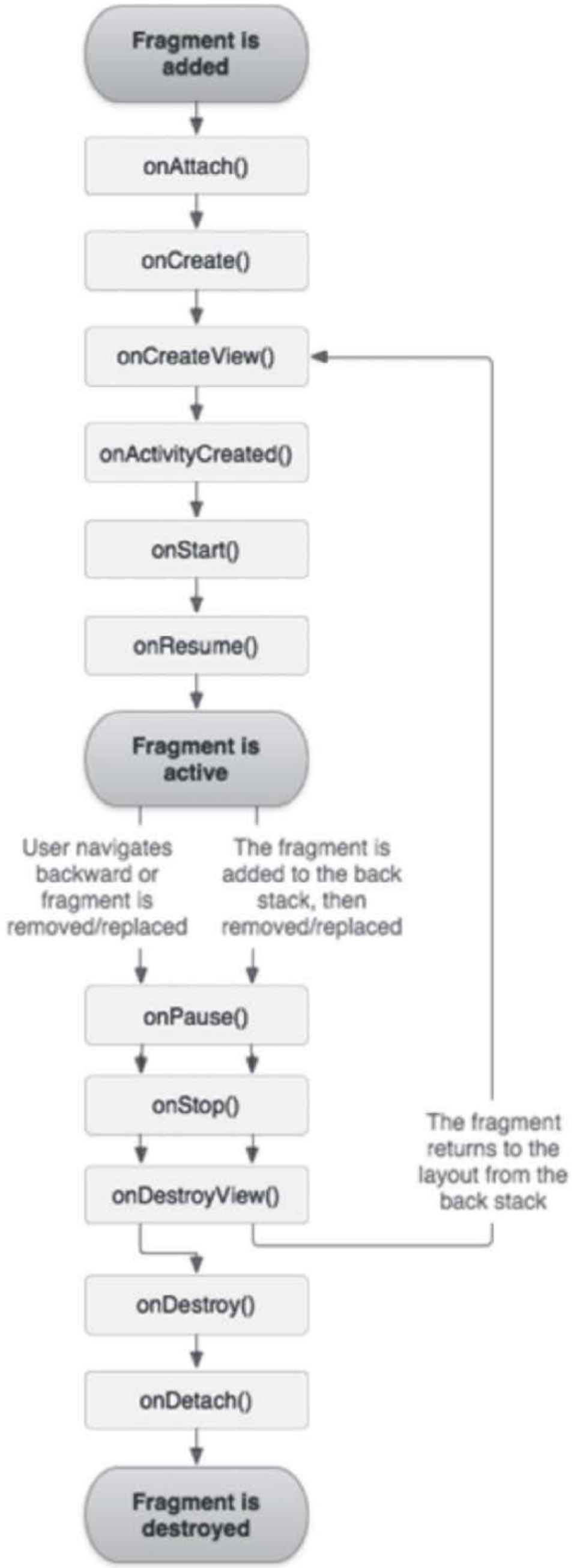
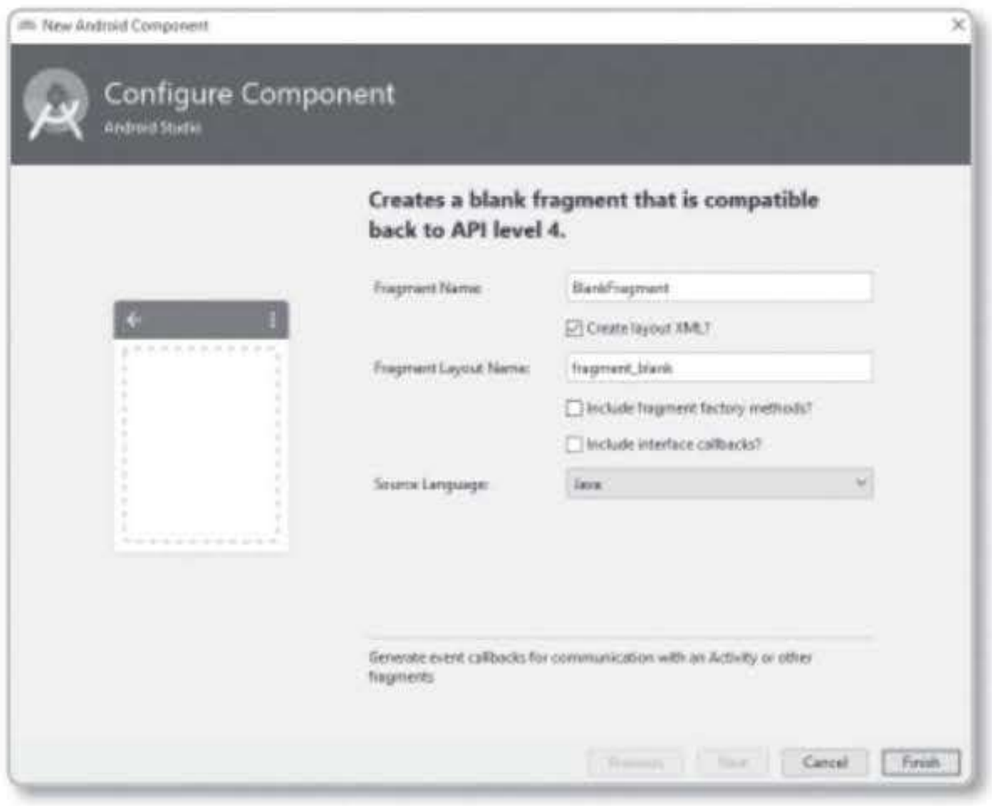


Figura 21. Fragmentos y eventos en Android. Tomado de "Fragmentos," por Developers, s.f. Recuperado de https://developer.android.com/guide/components/fragments?hl=es-419

3.10 Registrar fragmentos en la aplicación

En una aplicación, podemos agregar múltiples fragment que luego podrán ser usados en las distintas *activitys*. Para lograr esto, debemos asociar el archivo fragment con la activity. El fragment se comportará como una porción de la aplicación que se encuentra separada del elemento activity, pero unido por un medio de asociación. Para esto, debemos seguir los siguiente pasos:

- Hacemos clic derecho sobre la aplicación.
- Seleccionamos **New > Fragment > Fragment(Blank)**.
- Asignamos los datos correspondientes al fragment, como su nombre, y desactivamos **Inclide fragment factory methods?** e **Include interface callbacks**, tal como se muestra en la siguiente captura:



Veamos una aplicación en la que un título y un diseño dentro de un fragmento se muestre dentro de una activity. Primero, se debe agregar un fragment al proyecto y diseñar el siguiente modelo:



```
<?xml version='1.0' encoding='utf-8'?>
<FrameLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.PrincipalFragment'>

    <!-- TODO: Update blank fragment layout -->
    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:orientation='vertical'>

        <TextView
            android:id='@+id/textView'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:textColor='#000'
            android:gravity='center'
            android:textSize='30dp'
            android:layout_marginTop='30dp'
            android:text='BIBLIOTECA DIGITAL' />

        <ImageView
            android:id='@+id/imageView'
            android:layout_width='match_parent'
            android:layout_height='match_parent'
            android:layout_gravity='center'
            android:src='@drawable/logo' />
    </LinearLayout>
</FrameLayout>
```

Segundo, debemos agregar un control de tipo **<fragment>** en la activity y modificar el código de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>
    <fragment
        android:id='@+id/fragment'
        android:name='com.torres.manuel.myapplication.PrincipalFragment'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:layout_weight='1' />
</LinearLayout>
```

CASO DESARROLLADO 01: **asignar un ícono a nuestra aplicación**

Implementemos una aplicación en Android Studio que permita asignar un ícono al proyecto, el cual se mostrará al iniciar la aplicación en el dispositivo móvil.



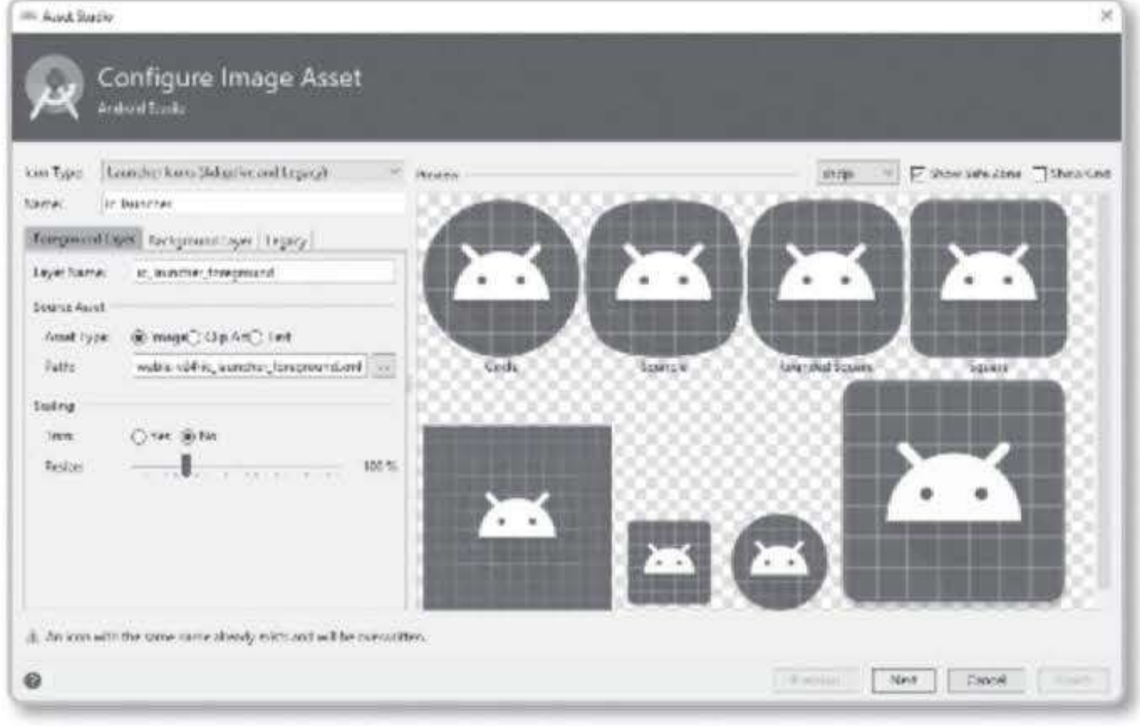
Restaurant.png

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 20):

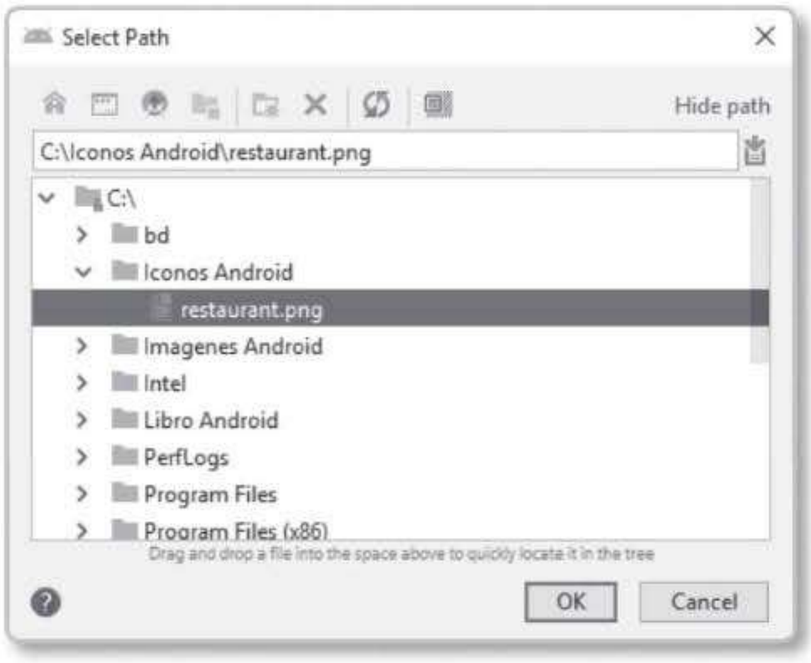
Tabla 20. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso31 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

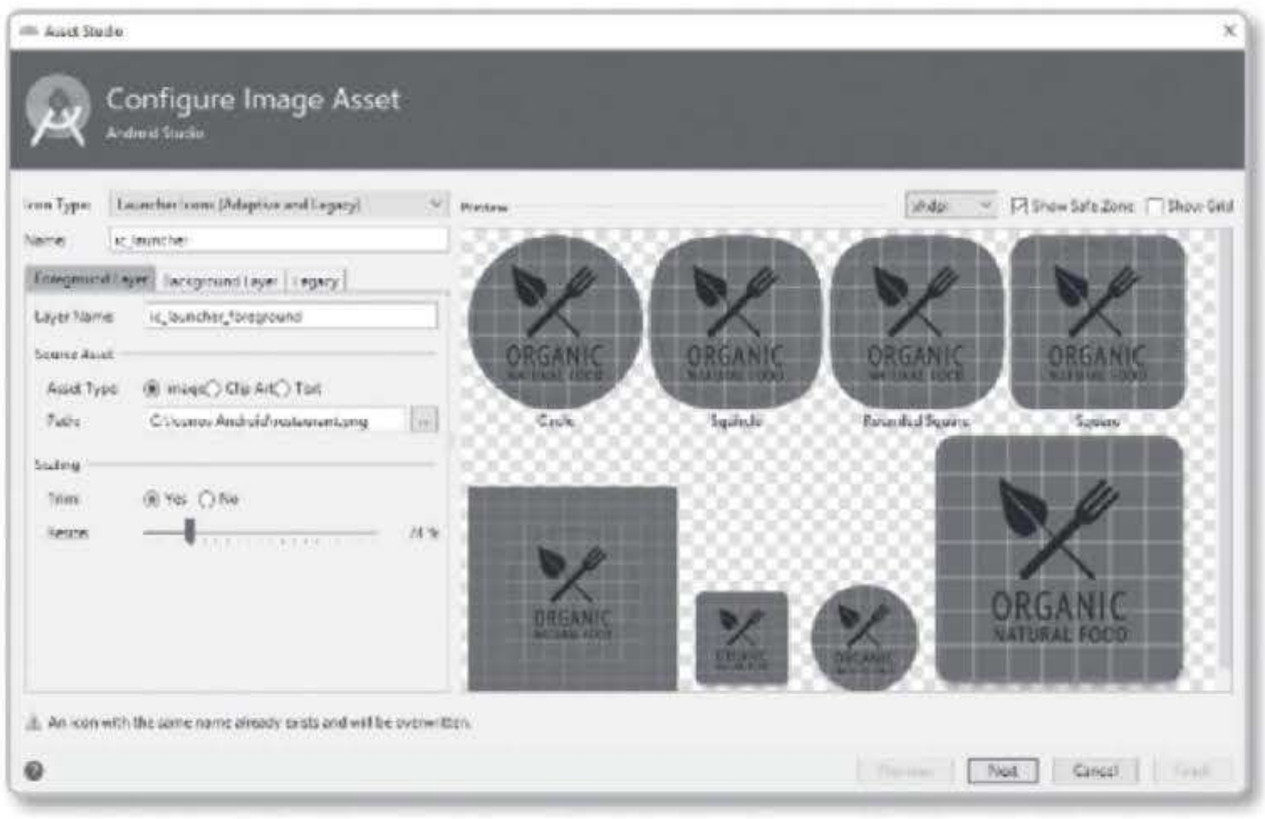
3. Agregamos las imágenes a la carpeta **mipmap** (clic derecho sobre la aplicación **app > New > Image Asset**), tal como se muestra en la siguiente captura:



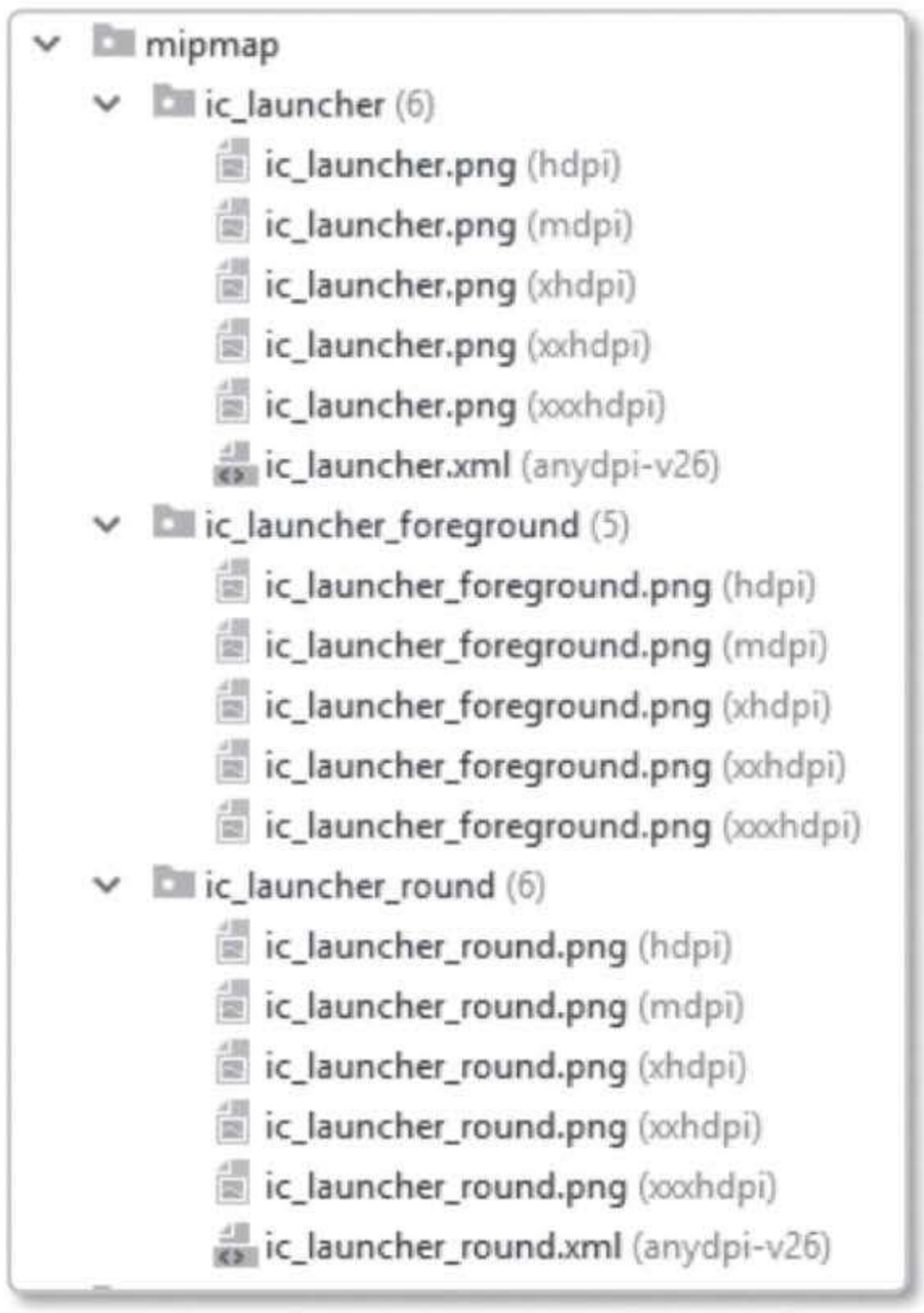
Luego, presionamos clic en restaurant.png de la opción **Path** para seleccionar una imagen de tipo png:



Luego de seleccionar la imagen, el entorno debe mostrarse de la siguiente manera:



Posteriormente, se crea un conjunto de imágenes que estarán distribuidas en la carpeta mipmap, tal como se muestra en la siguiente captura. Si desea visualizar el contenido de la imagen, será cuestión de darle doble clic en cualquier archivo con extensión .png.



4. El archivo **AndroidManifest.xml** se muestra de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android='http://schemas.android.com/apk/res/android'
    package='com.torres.manuel.pmcaso31'>

    <application
        android:allowBackup='true'
        android:icon='@mipmap/ic_launcher'
        android:label='@string/app_name'
        android:roundIcon='@mipmap/ic_launcher_round'
        android:supportsRtl='true'
        android:theme='@style/AppTheme'>
        <activity android:name='.MainActivity'>
            <intent-filter>
                <action android:name='android.intent.action.MAIN' />
              <category android:name='android.intent.category.LAUNCHER' />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Para modificar el título de la aplicación, podemos hacerlo presionando la tecla "CTRL+Clic" sobre **app_name** de la propiedad **android:label**.

```
<resources>
    <string name='app_name'>Manejo de Iconos</string>
</resources>
```

Finalmente, para modificar el texto por defecto que muestra la aplicación, debemos realizar las siguientes modificaciones al archivo **ActivityMain.xml**.

```
<?xml version='1.0' encoding='utf-8'?>
<RelativeLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>
    <TextView
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:text='LIBRO DE APLICACIONES ANDROID'
        android:layout_centerInParent='true'
         />
</RelativeLayout>
```

Ejecute la aplicación para ver los cambios, y si queremos visualizar el ícono, podemos observarlo desde el emulador móvil de la siguiente manera:



pjCaso31

CASO DESARROLLADO 02: asignar imagen de fondo a la aplicación

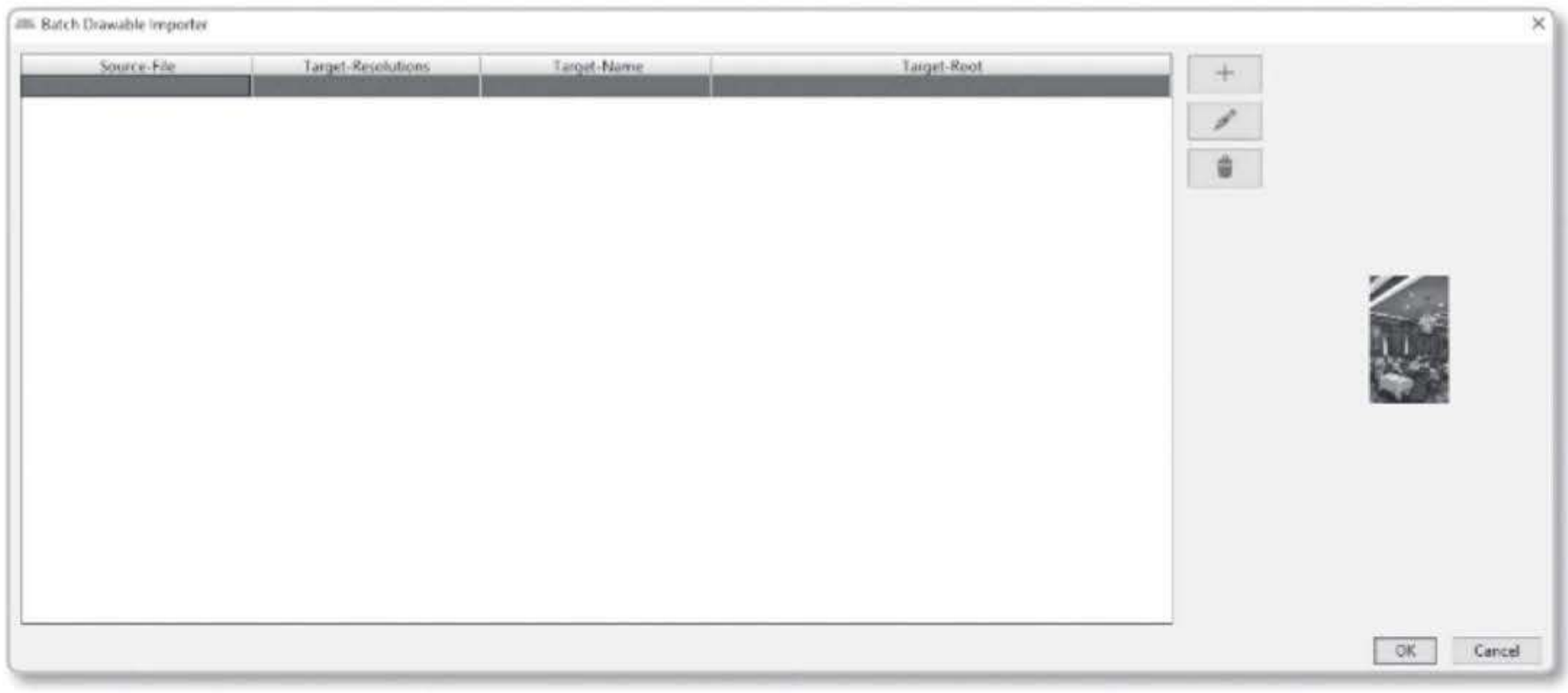
Implementemos una aplicación en Android Studio que permita asignar un título al proyecto (este se visualizará como ícono en su dispositivo móvil) y un encabezado dentro de la aplicación.

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 21):

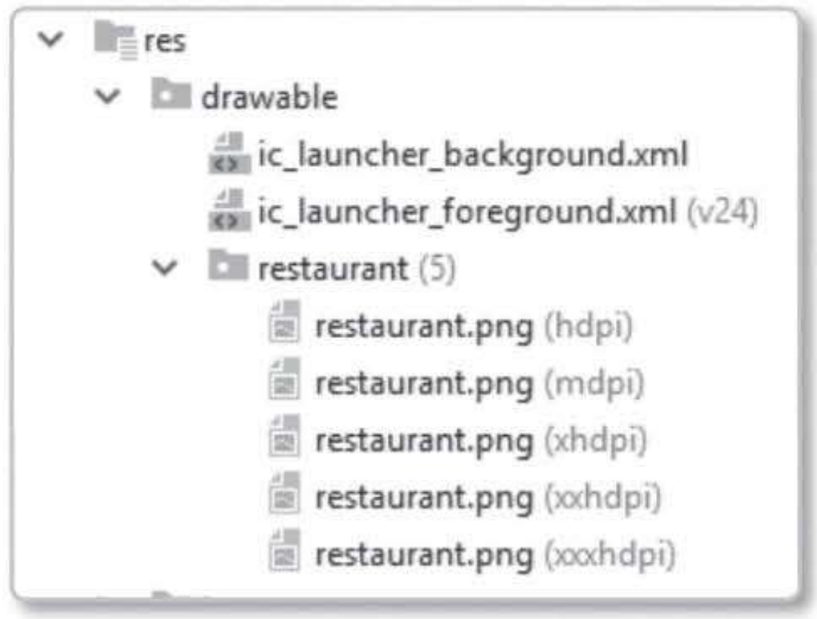
Tabla 21. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso32 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos la imagen de fondo a la carpeta **drawable** (clic derecho sobre la aplicación **app > New > Batch Drawable Import**), tal como se muestra a continuación.



Luego, presionamos clic en **+** para seleccionar la imagen y al final presione **OK**. Entonces, podremos visualizar la imagen seleccionada desde la carpeta **drawable**, tal como se muestra en la siguiente captura:



4. Modificamos el archivo **activity_main.xml** de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout xmlns:android='http://
schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:background='@drawable/restaurant'
    tools:context='.MainActivity'>
</android.support.constraint.ConstraintLayout>
```

Ejecutamos la aplicación para ver los cambios. Podemos observarlos desde el emulador móvil de la siguiente manera:



CASO DESARROLLADO 03: **asignar colores de fondo**

Implementemos una aplicación que permita modificar los colores del fondo de la pantalla de la aplicación por medio de tres botones.

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 22):

Tabla 22. Especificación de valores para el proyecto

Create Android Project	Application Name	: pmCaso33
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_03

Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Modificamos el archivo **strings.xml**, que se encuentra en la carpeta **values**, para especificar el título de la aplicación.

```
<resources>
    <string name='app_name'>Manejo de Colores</string>
</resources>
```

4. Modificamos el archivo **colors.xml**, que se encuentra en la carpeta **values**, para definir los colores blanco, rojo, verde y azul.

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
    <color name='colorPrimary'>#008577</color>
    <color name='colorPrimaryDark'>#00574B</color>
    <color name='colorAccent'>#D81B60</color>

    <color name='blanco'>#FFFFFF</color>
    <color name='rojo'>#FF0000</color>
    <color name='verde'>#00FF00</color>
    <color name='azul'>#0000FF</color>
</resources>
```

5. Agregamos tres botones dentro de un linear layout de tipo horizontal en el archivo **activity_main.xml**, tal como se muestra a continuación.

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <LinearLayout
        android:layout_width='368dp'
        android:layout_height='495dp'
        android:orientation='horizontal'
        tools:layout_editor_absoluteX='8dp'
        tools:layout_editor_absoluteY='8dp'
        tools:ignore='MissingConstraints'>

        <Button
            android:id='@+id/btnVerde'
            android:layout_width='fill_parent'
            android:layout_height='wrap_content'
            android:layout_weight='1'
            android:background='@color/verde'
            android:text='Verde' />
```

```
        <Button
            android:id="@+id/btnRojo"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@color/rojo"
            android:layout_weight="1"
            android:text="ROJO" />

        <Button
            android:id="@+id/btnAzul"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@color/azul"
            android:layout_weight="1"
            android:text="AZUL" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

6. Modificamos el archivo **MainActivity.java** para agregar el evento "clic sobre los botones". Debemos agregar la cláusula "implements OnClickListener" para habilitar el evento "clic de los botones".

```
package com.example.toshibapro.pjmanejocolors;

import android.support.v7.app.AppCompatActivity;
import android.view.View.OnClickListener;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity implements
OnClickListener {

    View view;
    Button btnRojo,btnVerde, btnAzul;

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

        //Definir el objeto de la clase View
        view = this.getWindow().getDecorView();
        view.setBackgroundResource(R.color.blanco);

        //Reconocer los controles tipo boton y asignar al listener
        btnRojo= (Button) findViewById(R.id.btnRojo);
        btnRojo.setOnClickListener(this);
        btnVerde= (Button) findViewById(R.id.btnVerde);
        btnVerde.setOnClickListener(this);
        btnAzul= (Button) findViewById(R.id.btnAzul);
        btnAzul.setOnClickListener(this);
    }

    //Metodo que permite accionar de acuerdo al boton seleccionado
    public void onClick(View v){
      if(v.getId()==R.id.btnRojo) view.setBackgroundResource(R.color.rojo);
```

```
    if(v.getId()==R.id.btnVerde) view.setBackgroundResource(R.color.verde);
    if(v.getId()==R.id.btnAzul) view.setBackgroundResource(R.color.azul);
        }
    }
```

7. Ejecutamos la aplicación para ver los cambios. Podemos apreciarlos desde el emulador móvil de la siguiente manera:



CASO DESARROLLADO 04: **asignar títulos a la aplicación**

Implementemos una aplicación en Android Studio que permita asignar un título al proyecto (este se visualizará como ícono en su dispositivo móvil) y un encabezado dentro de la aplicación.

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 23):

Tabla 23. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso34 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Definimos las cadenas en el archivo **strings.xml**.

```
    <resources>
        <string name="app_name">Restaurant</string>
        <string name="titulo">SISTEMA DE GESTIÓN - RESTAURANT</string>
    </resources>
```

4. Modificamos el archivo **AndroidManifest.xml** para asignar el título a la pantalla inicial.

```
    <?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.torres.manuel.pmcaso34">
        <application
```

```
        android:allowBackup='true'
        android:icon='@mipmap/ic_launcher'
        android:label='@string/app_name'
        android:roundIcon='@mipmap/ic_launcher_round'
        android:supportsRtl='true'
        android:theme='@style/AppTheme'>
        <activity android:name='.MainActivity'>
            <intent-filter>
                <action android:name='android.intent.action.MAIN' />
                <category android:name='android.intent.category.LAUNCHER' />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

5. Modificamos el archivo **activity_main.xml** para asignar el título a la pantalla inicial.

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>
    <TextView
        android:id='@+id/textView2'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:text='@string/titulo'
        tools:layout_editor_absoluteX='16dp'
        tools:layout_editor_absoluteY='16dp'
        tools:ignore='MissingConstraints' />
</android.support.constraint.ConstraintLayout>
```

6. Ejecutamos la aplicación para ver los cambios. Podemos observarlos desde el emulador móvil de la siguiente manera:



CASO DESARROLLADO 05: **manejo de arreglos de textos**

Implementemos una aplicación móvil en Android que permita mostrar una lista de productos y sus precios a partir del uso de arreglos.

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 24):

Tabla 24. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso35 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Definimos los arreglos en el archivo **strings.xml**, que se encuentra en la carpeta **values**.

```
<resources>
    <string name="app_name">Manejo de Arreglos de Textos</string>

    <!--Definicion de descripcion de productos-->
    <string-array name="aProductos">
        <item>Ceviche</item>
        <item>Seco de cabrito</item>
        <item>Ocopa</item>
        <item>Pachamanca</item>
        <item>Anticucho</item>
    </string-array>

    <!--Definicion de precios por producto-->
    <string-array name="aPrecios">
        <item>S/30.00</item>
        <item>S/35.00</item>
        <item>S/15.00</item>
        <item>S/45.00</item>
        <item>S/25.00</item>
    </string-array>
</resources>
```

4. Modificamos el archivo **activity_main.xml** para mostrar un linearlayout con orientación vertical y, dentro de este, colocamos un texto y un nuevo linearlayout de orientación horizontal para, finalmente, agregar los dos ListView que tendrán la misión de mostrar la información de los productos y sus respectivos precios.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

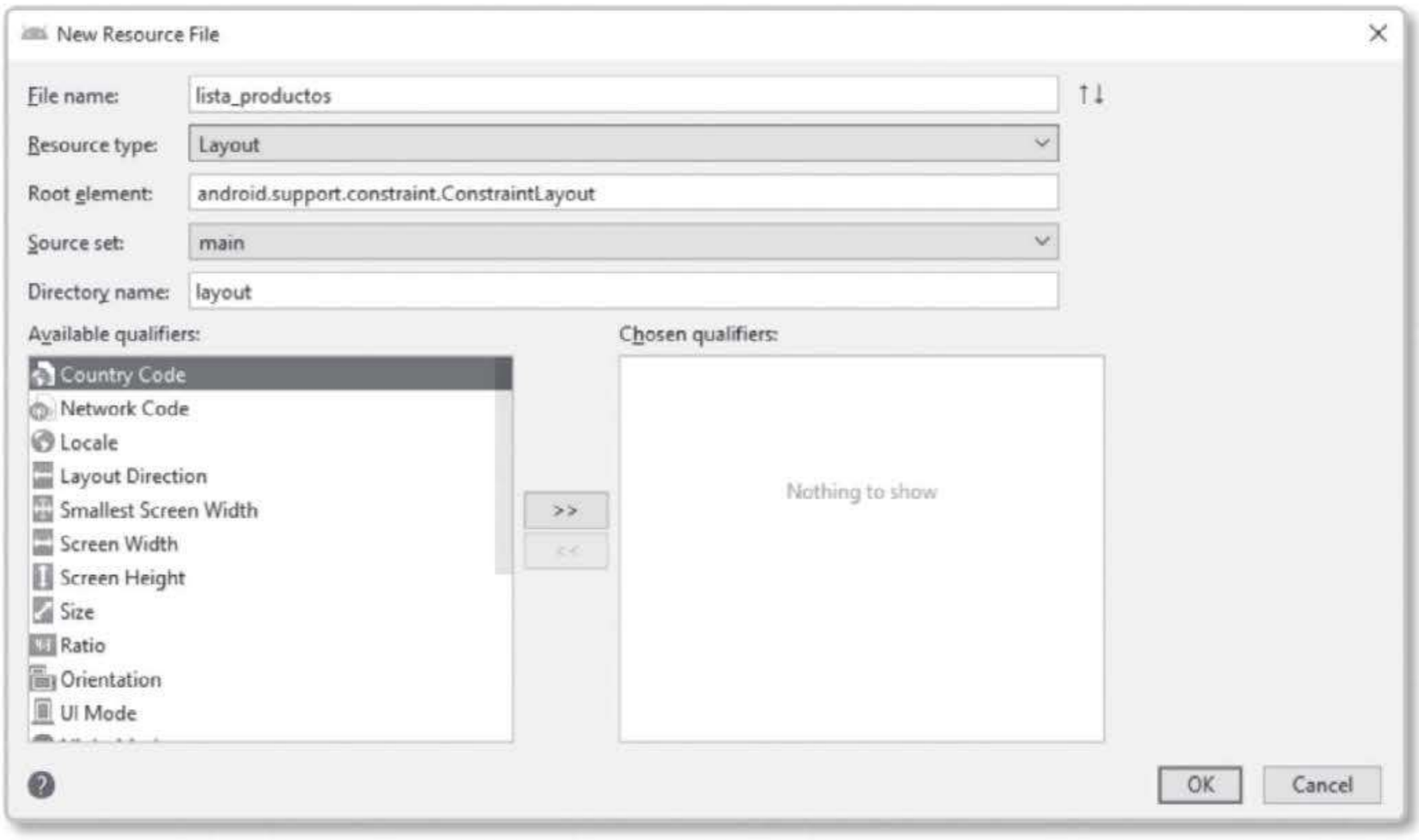
    <LinearLayout
        android:layout_width="368dp"
        android:layout_height="495dp"
        android:orientation="vertical"
        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="8dp"
        tools:ignore="MissingConstraints">
```

```
        <TextView
            android:id="@+id/textView2"
            android:layout_width="375dp"
            android:layout_height="49dp"
            android:layout_marginEnd="8dp"
            android:gravity="center"
            android:text="Listado de Productos - Arrays"
            android:textColor="#000"
            android:textSize="20dp"
            app:layout_constraintEnd_toEndOf="parent"
            tools:ignore="MissingConstraints"
            tools:layout_editor_absoluteY="1dp" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal">

            <ListView
                android:id="@+id/lvProductos"
                android:layout_width="195dp"
                android:layout_height="wrap_content"
                android:divider="#000"
                android:dividerHeight="2dp"
                tools:ignore="MissingConstraints"
                tools:layout_editor_absoluteX="-2dp"
                tools:layout_editor_absoluteY="34dp" />
            <ListView
                android:id="@+id/lvPrecios"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:divider="#000"
                android:dividerHeight="2dp"
                tools:ignore="MissingConstraints"
                tools:layout_editor_absoluteX="-2dp"
                tools:layout_editor_absoluteY="34dp" />
        </LinearLayout>
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

5. Agregamos el archivo **lista_productos.xml** de tipo Layout para agregar un control TextView, el cual contendrá los nombres de los productos. Para ello, se debe hacer clic derecho sobre el **proyecto > New > Android Resource File**.




```
<?xml version='1.0' encoding='utf-8'?>
<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:id='@+id/tvProductos'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='TextView'
        android:textColor='#000'
        />
</LinearLayout>
```

6. Agregamos el archivo **lista_precios.xml** de tipo Layout para agregar un control TextView, el cual contendrá los precios de los productos (clic derecho sobre el **proyecto > New > Android Resource File**).

```
<?xml version='1.0' encoding='utf-8'?>
<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:id='@+id/tvPrecios'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:textColor='#000'
        android:text='TextView' />
</LinearLayout>
```

7. Modificamos el archivo **MainActivity.java** para asignar los productos y sus precios a los controles ListView.

```
package com.torres.manuel.pmcaso35;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

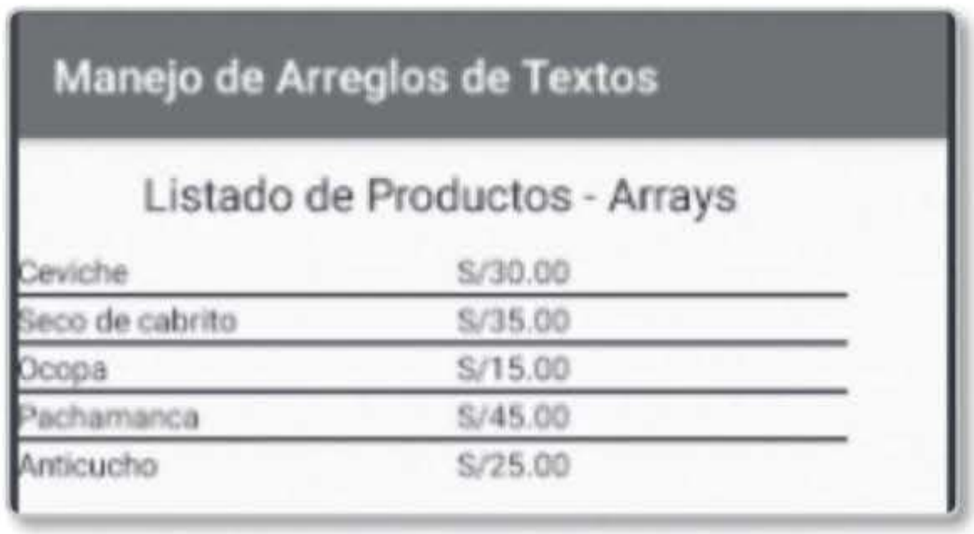
    //Definicion de los controles
    ListView lvProductos, lvPrecios;

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

        lvProductos=(ListView)findViewById(R.id.lvProductos);
        lvPrecios=(ListView)findViewById(R.id.lvPrecios);

       ArrayAdapter<String> adapterProductos=new ArrayAdapter<String>(this,
                                R.layout.lista_productos,R.id.tvProductos,
                            getResources().getStringArray(R.array.aProductos));
        ArrayAdapter<String> adapterPrecios=new ArrayAdapter<String>(this,
                                 R.layout.lista_precios,R.id.tvPrecios,
                             getResources().getStringArray(R.array.aPrecios));
            lvProductos.setAdapter(adapterProductos);
            lvPrecios.setAdapter(adapterPrecios);
        }
    }
```

8. Ejecutamos la aplicación para ver los cambios. Podemos observarlos desde el emulador móvil de la siguiente manera:



pmCaso35

CASO DESARROLLADO 06: **asignación de estilos**

Implementemos una aplicación móvil en Android que permita mostrar información de un plato típico del Perú usando estilos en sus controles.

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 25):

Tabla 25. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso36 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Definimos los estilos en el archivo **styles.xml** que se encuentra en la carpeta values.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="EstiloTexto">
        <item name="android:textColor">#000000</item>
        <item name="android:textSize">20dp</item>
        <item name="android:fontFamily">tahoma</item>
    </style>

    <style name="EstiloBoton">
        <item name="android:drawableLeft">@drawable/peru </item>
        <item name="android:textSize">25dp</item>
        <item name="android:padding">15dp</item>
    </style>

    <style name="EstiloBoton.rojo">
        <item name="android:background">#FF0000</item>
        <item name="android:textColor">#FFFFFF</item>
    </style>
    <style name="EstiloBoton.verde">
        <item name="android:background">#00FF00</item>
        <item name="android:textColor">#FFFFFF</item>
    </style>
    <style name="EstiloBoton.azul">
        <item name="android:background">#0000FF</item>
        <item name="android:textColor">#FFFFFF</item>
    </style>
</resources>
```

4. Modificamos el archivo **activity_main.xml** para mostrar un linearlayout con orientación vertical y, dentro de este, colocamos un texto y un nuevo linearlayout de orientación horizontal para, finalmente, agregar los dos ListView que tendrán la misión de mostrar la información de los productos y sus respectivos precios.

```
<?xml version='1.0' encoding='utf-8'?>
<RelativeLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <android.support.v7.widget.Toolbar
        android:id='@+id/toolbar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:background='#0000FF'
        app:titleTextColor='#FFFFFF'
        app:subtitleTextColor='#F0FF0F'
        android:elevation='4dp'
        android:minHeight='?attr/actionBarSize'
        />

    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:layout_alignParentStart='true'
        android:layout_alignParentTop='true'
        android:layout_marginStart='0dp'
        android:layout_marginTop='0dp'
        android:orientation='vertical'>

        <ImageView
            android:id='@+id/imageView'
            android:layout_width='match_parent'
            android:layout_height='254dp'
            app:srcCompat='@drawable/ceviche' />

        <EditText
            android:id='@+id/editText'
            android:layout_width='match_parent'
            android:layout_height='138dp'
            android:ems='10'
            android:inputType='textMultiLine'
            android:padding='10dp'
            style='@style/EstiloTexto'
            android:text='El ceviche es un plato típico de la gastronomía
          Peruana, su preparación se basa en pescado, limón y el secreto que
            solo se puede dar en el Perú.' />

        <Button
            android:id='@+id/button3'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            style='@style/EstiloBoton.rojo'
            android:text='MAS INFORMACIÓN' />

        <Button
            android:id='@+id/button4'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            style='@style/EstiloBoton.verde'
            android:text='VER VIDEO' />
```

```
            <Button
                android:id='@+id/button5'
                android:layout_width='match_parent'
                android:layout_height='wrap_content'
                style='@style/EstiloBoton.azul'
                android:text='SIGUIENTE PLATO' />
        </LinearLayout>
    </RelativeLayout>
```

5. Modificamos el archivo **MainActivity.java**.

```
package com.torres.manuel.pmcaso36;

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

public class MainActivity extends AppCompatActivity {

    Toolbar toolbar;

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

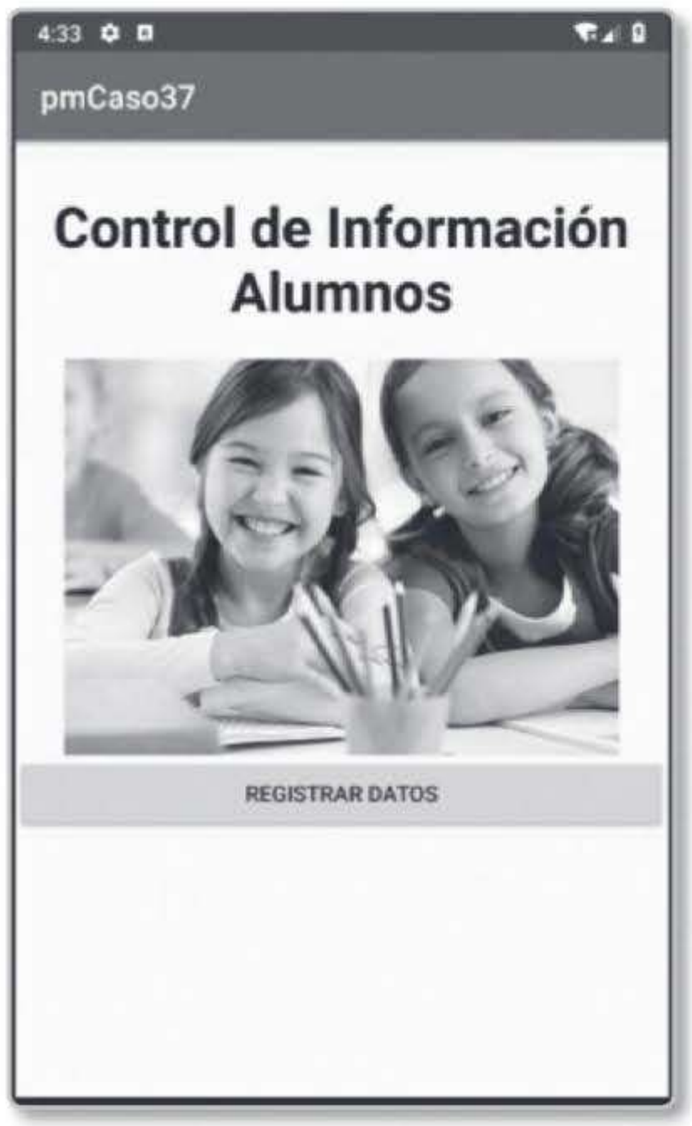
        toolbar = (Toolbar)findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        toolbar.setTitle('Cocina Familiar');
        toolbar.setSubtitle('Ceviche Peruano');
    }
}
```

6. Ejecutamos la aplicación para ver los cambios. Podemos observarlo desde el emulador móvil de la siguiente manera:



CASO DESARROLLADO 07: **manejo de actividades**

Implementemos una aplicación móvil que permita registrar información de los alumnos de una determinada escuela. Inicialmente, la aplicación debe mostrar la pantalla que se expone en la captura que se aprecia a continuación. Asimismo, para acceder al registro, deberá seleccionar el botón **Registrar Datos**.



Al ingresar a la actividad "Registro de Alumnos", se debe mostrar de la siguiente manera:



Finalmente, al llenar la información, se debe presionar el botón **Registrar** para enviar los datos a una actividad de información que muestre los datos ingresados, tal como se muestra a continuación.



1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 26):

Tabla 26. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso37 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_03
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Modificamos el archivo **activity_main.xml** para obtener la pantalla inicial de la aplicación.

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>
    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:layout_marginBottom='20dp'
```

```
        android:layout_marginTop='30dp'
        android:textStyle='bold'
        android:textSize='35dp'
        android:textColor='#000'
        android:gravity='center'
        android:text='Control de Información Alumnos' />

    <ImageView
        android:id='@+id/imageView'
        android:layout_width='match_parent'
        android:layout_height='250dp'
        app:srcCompat='@drawable/educacion' />

    <Button
        android:id='@+id/btEnviar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='REGISTRAR DATOS' />
</LinearLayout>
```

4. Agregamos un activity al proyecto y lo nombramos **registro.xml**.

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.registro'>

    <TextView
        android:id='@+id/textView2'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:layout_marginBottom='30dp'
        android:layout_marginTop='30dp'
        android:textSize='25dp'
        android:textColor='#000'
        android:gravity='center'
        android:textStyle='bold'
        android:text='REGISTRO DE ALUMNOS' />

    <EditText
        android:id='@+id/etNombres'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:ems='10'
        android:hint='NOMBRES'
        android:inputType='textPersonName' />

    <EditText
        android:id='@+id/etApellidos'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:ems='10'
        android:hint='APELLIDOS'
        android:inputType='textPersonName' />
```

```
        <EditText
            android:id="@+id/etCorreo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="CORREO"
            android:inputType="textEmailAddress" />

        <EditText
            android:id="@+id/etTelefono"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="TELEFONO"
            android:inputType="phone" />

        <EditText
            android:id="@+id/etComentarios"
            android:layout_width="match_parent"
            android:layout_height="147dp"
            android:ems="10"
            android:hint="COMENTARIOS"
            android:inputType="textMultiLine" />

        <Button
            android:id="@+id/btRegistrar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="REGISTRAR" />
    </LinearLayout>
```

- Agregamos un activity al proyecto y lo nombramos **informacion.xml**.

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        tools:context=".informacion">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <TextView
                android:id="@+id/textView5"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="10dp"
                android:text="NOMBRES" />

            <TextView
                android:id="@+id/textView4"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="10dp"
                android:text="APELLIDOS" />
```

```
        <TextView
            android:id="@+id/textView3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="CORREO" />

        <TextView
            android:id="@+id/textView6"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="TELEFONO" />

        <TextView
            android:id="@+id/textView7"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="COMENTARIOS" />

        <Button
            android:id="@+id/btnRetornar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="RETORNAR" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tvNombres"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="TextView" />

        <TextView
            android:id="@+id/tvApellidos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="TextView" />

        <TextView
            android:id="@+id/tvCorreo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="TextView" />

        <TextView
            android:id="@+id/tvTelefono"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="TextView" />
```

```
            <TextView
                android:id="@+id/tvComentarios"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="10dp"
                android:text="TextView" />
        </LinearLayout>
    </LinearLayout>
```

5. Modificamos el archivo **MainActivity.java**.

```
package com.torres.manuel.pmcaso37;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        Button btEnviar=(Button) findViewById(R.id.btEnviar);
        btEnviar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent(getApplication(),registro.class);
                 startActivity(intent);
            }
        });
    }
}
```

6. Modificamos el archivo **registro.java**. Este se encuentra en nuestro proyecto, en la sección Java, y fue creado al agregar un activity.

```
package com.torres.manuel.pmcaso37;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class registro extends AppCompatActivity {

    Button btRegistrar;

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

```
        //Obtener los valores del formulario
        final EditText etNombres=(EditText) findViewById(R.id.etNombres);
        final EditText etApellidos=(EditText) findViewById(R.id.etApellidos);
        final EditText etCorreo=(EditText) findViewById(R.id.etCorreo);
        final EditText etTelefono=(EditText) findViewById(R.id.etTelefono);
        final EditText etComentarios=(EditText) findViewById(R.id.etComentarios);

        btRegistrar = (Button)findViewById(R.id.btRegistrar);
        btRegistrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Invocar al formulario informacion por medio de Intent
                Intent intent=new Intent(getApplication(),informacion.class);

                //Enviando parametros a la activity informacion.class
                intent.putExtra('nombres',etNombres.getText().toString());
                intent.putExtra('apellidos',etApellidos.getText().
toString());
                intent.putExtra('correo',etCorreo.getText().toString());
                intent.putExtra('telefono',etTelefono.getText().toString());
                intent.putExtra('comentarios',etComentarios.getText().
toString());

                //Iniciar la activity
                startActivity(intent);
            }
        });
    }
}
```

7. Modificamos el archivo **informacion.java**.

```
package com.torres.manuel.pmcaso37;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class informacion extends AppCompatActivity {

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

        Button btnRetornar=(Button) findViewById(R.id.btnRetornar);

        TextView tvNombres=(TextView) findViewById(R.id.tvNombres);
        TextView tvApellidos=(TextView) findViewById(R.id.tvApellidos);
        TextView tvCorreo=(TextView) findViewById(R.id.tvCorreo);
        TextView tvTelefono=(TextView) findViewById(R.id.tvTelefono);
        TextView tvComentarios=(TextView) findViewById(R.id.tvComentarios);

        //Obtener los parametros que fueron enviados desde registro.java
        Bundle datos=getIntent().getExtras();
```

```
        String nombres=datos.getString('nombres');
        String apellidos=datos.getString('apellidos');
        String correo=datos.getString('correo');
        String telefono=datos.getString('telefono');
        String comentario=datos.getString('comentarios');

        //Asignar los valores obtenidos a los controles de informacion
        tvNombres.setText(nombres);
        tvApellidos.setText(apellidos);
        tvCorreo.setText(correo);
        tvTelefono.setText(telefono);
        tvComentarios.setText(comentario);

        //Codigo del boton de retorno a la primera activity
        btnRetornar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              Intent intent=new Intent(getApplication(),MainActivity.class);
                startActivity(intent);
            }
        });

    }
}
```

CAPÍTULO

4



Controles básicos y notificaciones para aplicaciones Android

4.1 Interfaces de usuario

Una interfaz de usuario es considerada como un medio de interacción entre el usuario y el sistema móvil. Esto se debe a que por la pantalla podrá visualizar resultados de solicitudes que realiza el usuario y esta debe presentarse de manera armoniosa tanto en colores, distribución de elementos, entre otros aspectos. A continuación, observamos la jerarquía de clases entre los View y los ViewGroup (ver Figura 22).

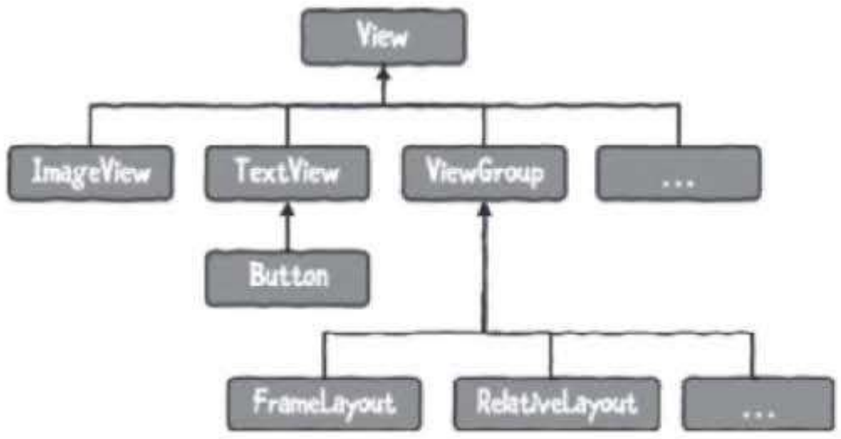


Figura 22. Jerarquía de Clase Android Tomado de "Android Custom View Tutorial," por A. Tarek, 2018. Recuperado de https://www.raywenderlich.com/142-android-custom-view-tutorial

Por otro lado, la interfaz de usuario es también llamada IU y se construye a partir de la jerarquía de diseños provenientes de la clase ViewGroup, los cuales controlan el posicionamiento de los controles en pantalla, y los Widgets, que son los componentes que complementan la interfaz por medio de botones, cuadros de textos, imágenes, etc. Para comprender cómo funciona la interfaz gráfica, debemos tener en claro los conceptos de View y ViewGroup.

- **View (vista)**: Es la plataforma contenedora de componentes visuales de una aplicación móvil. Una vista ocupa un área rectangular en la pantalla del dispositivo y controla los eventos que pueden tener los componentes (ver Figura 23).

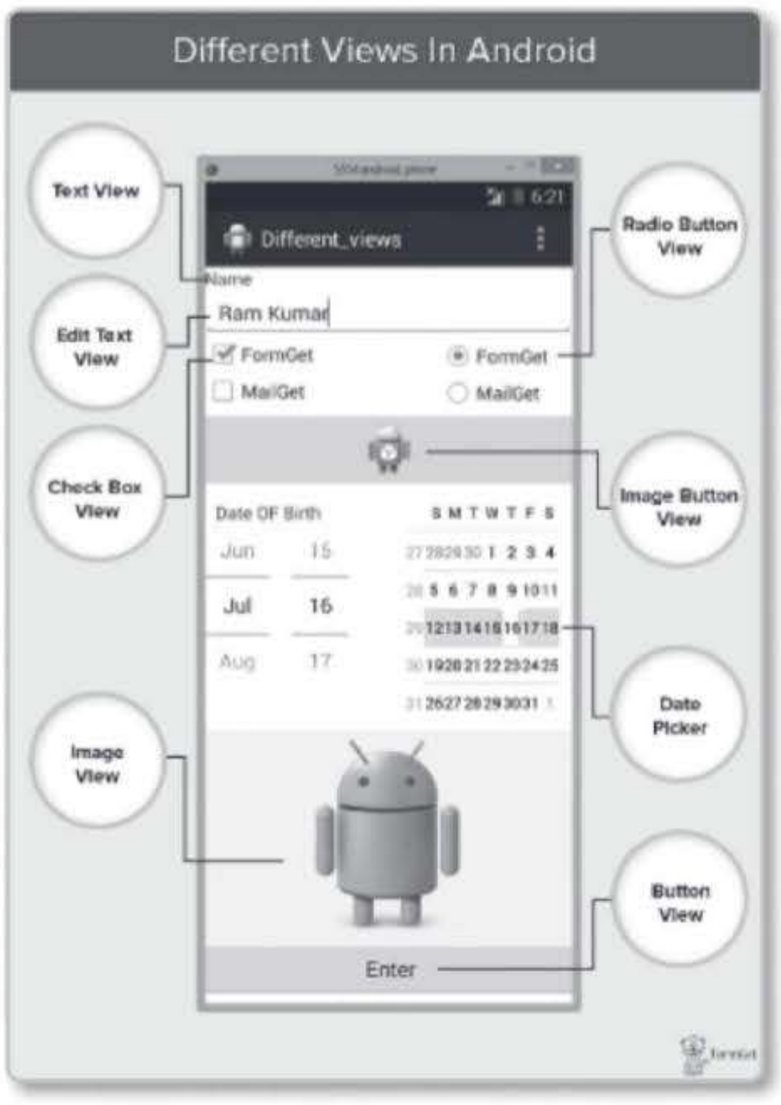
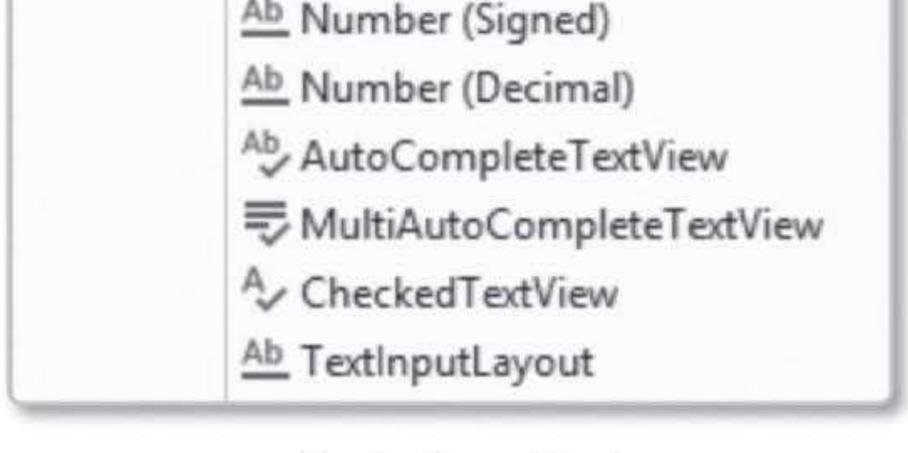


Figura 23. Diferentes vistas en Android. Tomado de "Android View Classes," por N. Agarwal, 2018. Recuperado de https://www.formget.com/android-views/.

- **ViewGroup (grupo de vista):** Es un objeto no visible para el usuario y se usa como contenedor de otros objetos de tipo View u otros ViewGroup. Tiene por finalidad organizar y controlar el layout de una pantalla en un aplicación móvil (ver Figura 24).

Controles de texto



Controles widgets

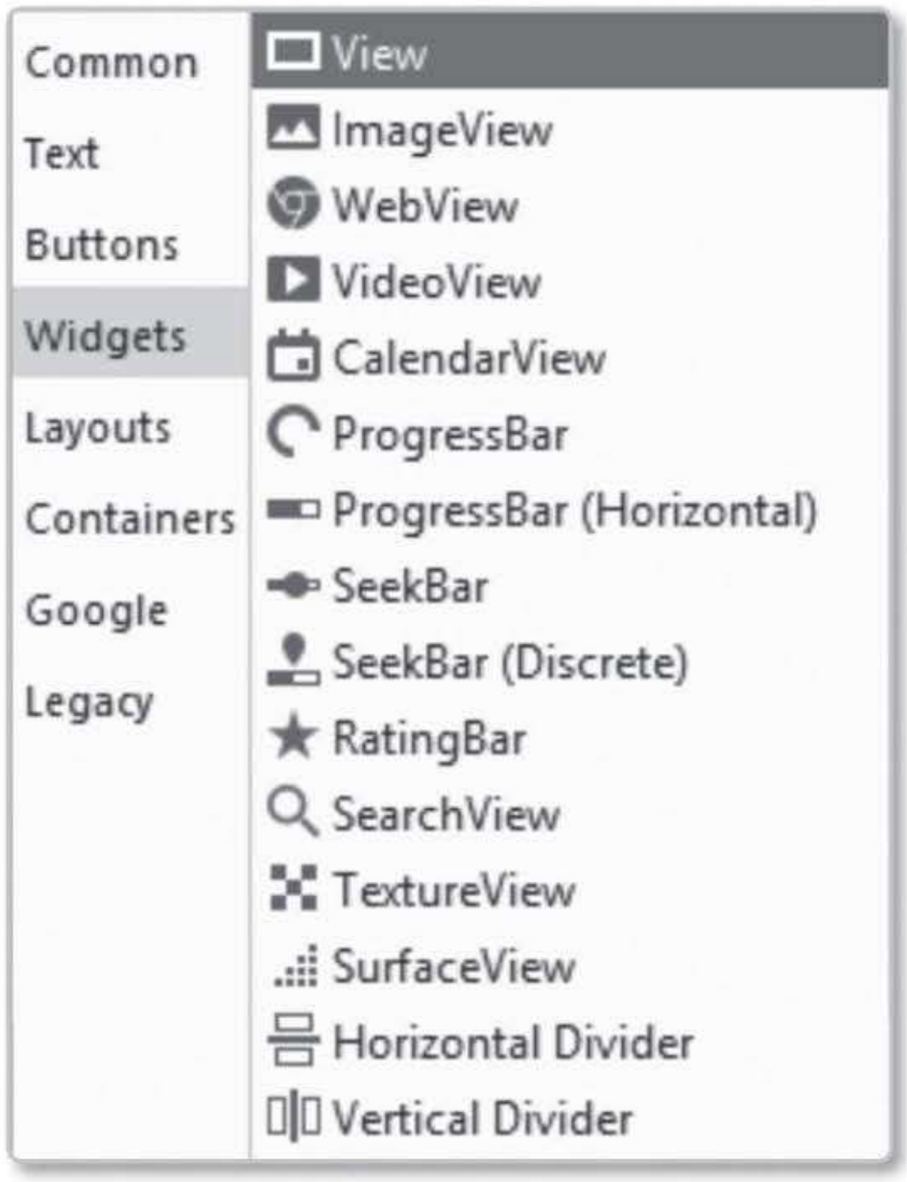




Figura 24. Controles Android. Tomado de "Palette.pngw 806 x 622 pixeles" y "Elementos y diseño de Interfaz de Usuario en Apps Android," por Digital Learning SL, s.f. Recuperado, respectivamente, de https://academiaandroid.com/wpaa/wp-content/uploads/2016/05/Palette.png y https://academiaandroid.com/elementos-y-diseno-de-interfaz-de-usuario-en-apps-android/

4.2 Densidad y tamaño de la pantalla

La densidad de píxeles se refiere a la cantidad de estos que pueden encajar en una pulgada. Debemos tener en cuenta que los dispositivos móviles vienen en variados tamaños; por tanto, esta variación hace que los componentes del diseño no siempre se vean bien en todos los dispositivos. Por ejemplo, las pantallas de baja densidad aparecen grandes, mientras que se muestran pequeñas en dispositivos de alta densidad.

En Android Studio, se muestran los posibles dispositivos en los que se ejecutará nuestra aplicación móvil (**Tool>AVD Manager**). Desde aquí podemos seleccionar dicho emulador y así visualizar el aspecto que tendrá al ejecutarlo.



Tamaños de pantalla. Es el espacio visible proporcionado por la UI de una aplicación móvil. Debemos considerar que dicho tamaño no es el espacio real proporcionado por el dispositivo, pues debemos descontar la orientación de la pantalla, las barras de navegación, etc. (ver Figura 25).

De forma intuitiva, Android modifica el tamaño del diseño de la aplicación móvil para ajustarlo al dispositivo móvil donde se ejecuta la aplicación. Por lo tanto, la recomendación es no asignar tamaños exactos a los componentes de esta, es decir, mantener la flexibilidad que nos ofrece Android mediante sus múltiples propiedades. Por ejemplo, algo que no debemos realizar es asignar una ubicación exacta a un control dentro de la pantalla, pues no sabemos en qué tipos de dispositivo se ejecutará.

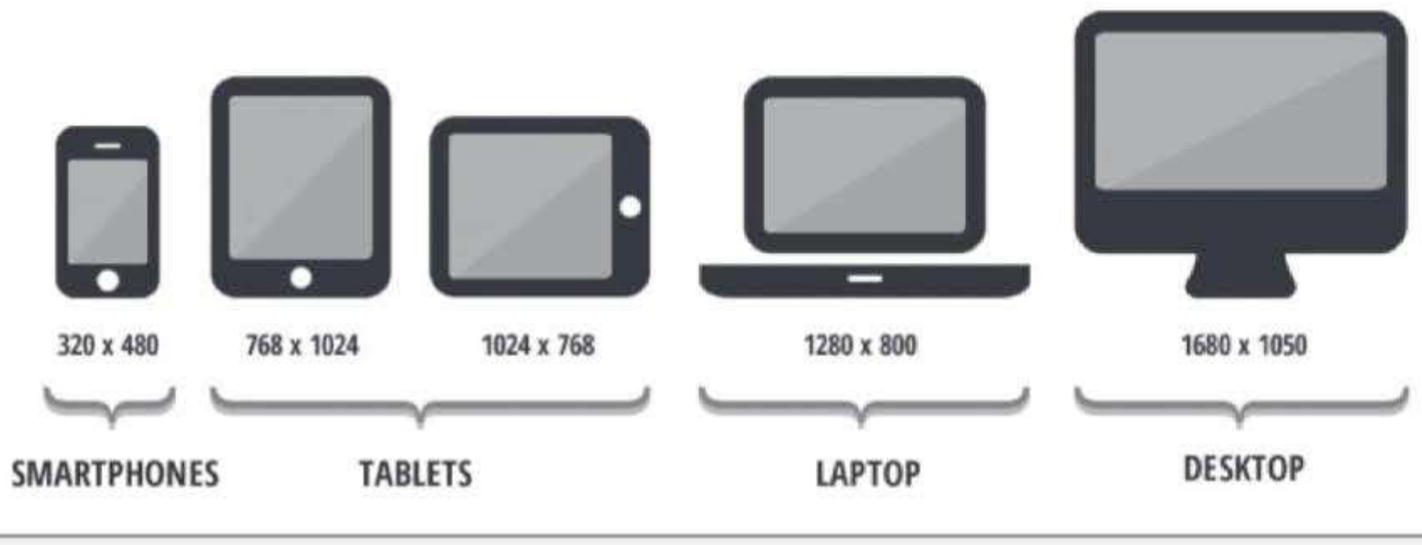


Figura 25. Tamaños de pantalla. Tomado de "El uso de dispositivos móviles crece muy rápidamente La importancia de los usuarios," por Weweb, s.f. Recuperado de http://www.websresponsivas.com/

4.3 Clase View

Las vistas son elementos que componen la interfaz de usuario de una aplicación: podría ser un botón, una lista o una imagen. Todas las vistas son consideradas como objetos descendientes de la clase View, los cuales pueden ser implementados mediante script XML o código Java (ver Figura 26). El caso

podría presentarse cuando se crea dinámicamente controles por medio de datos no exactos, por ejemplo, al crear cuadros de tipo check de acuerdo con una determinada condición de la aplicación.

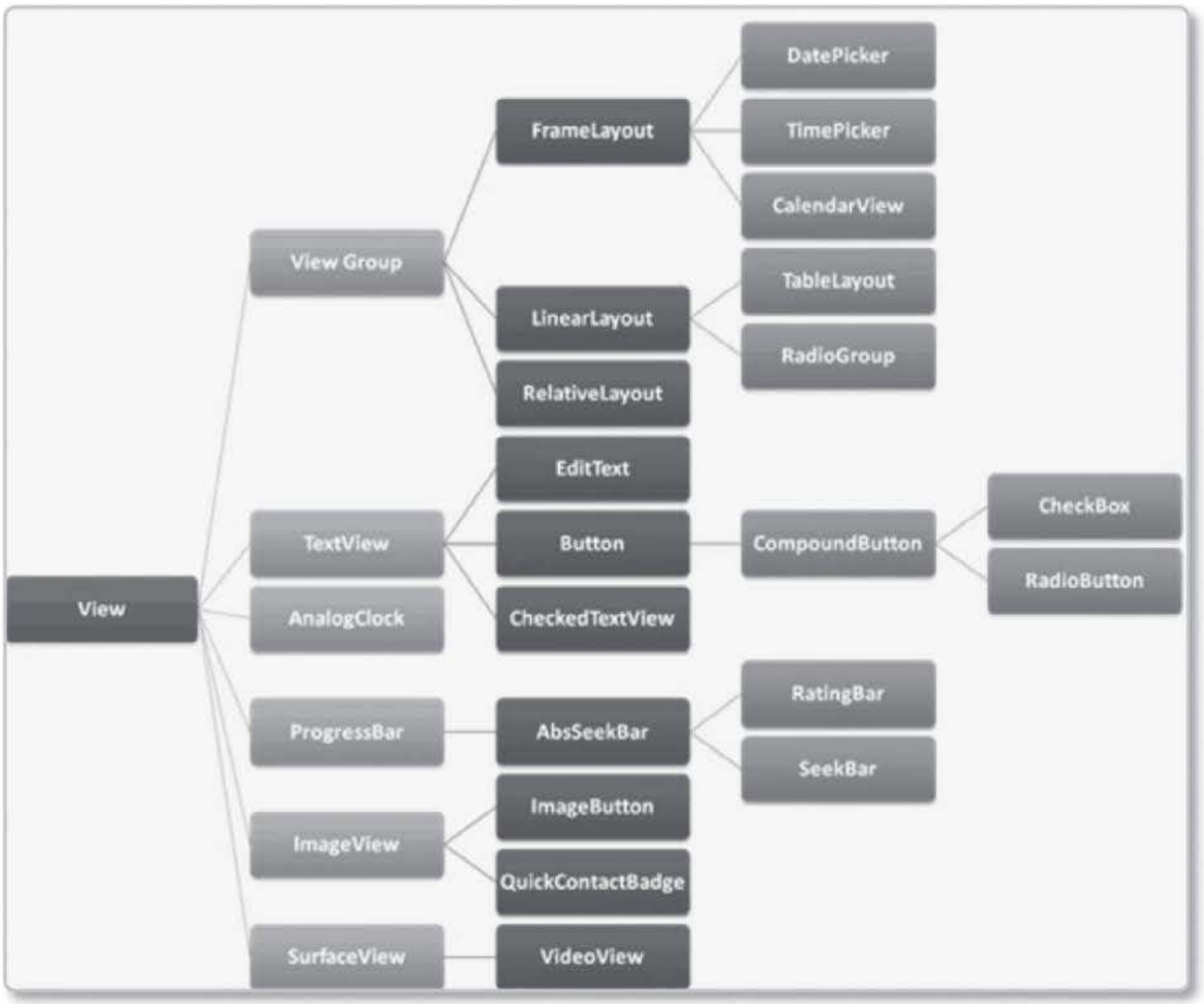


Figura 26. Jerarquía de la clase View. Tomado de "Conceptos Generales Android. Archivo AndroidManifest.xml," por Aprende y comparte. Tema fantástico, 2013. Recuperado de http://androiddummies.blogspot.com/2013/06/conceptos-generales-android-archivo.html

4.4 Distribución de los ViewGroups

Los layouts son elementos de una aplicación móvil que no son visibles por el usuario. Estos controlan la distribución, el posicionamiento y las dimensiones de todos los controles que se pueden alojar dentro de un grupo de vistas. Su capacidad es ilimitada. Ya que heredan de la clase ViewGroup la cantidad de componentes, será la mejor distribución que el usuario quiera darle a la aplicación.

En la Tabla 27, se aprecia una lista de las principales propiedades de tipo XML para los controles en Android.

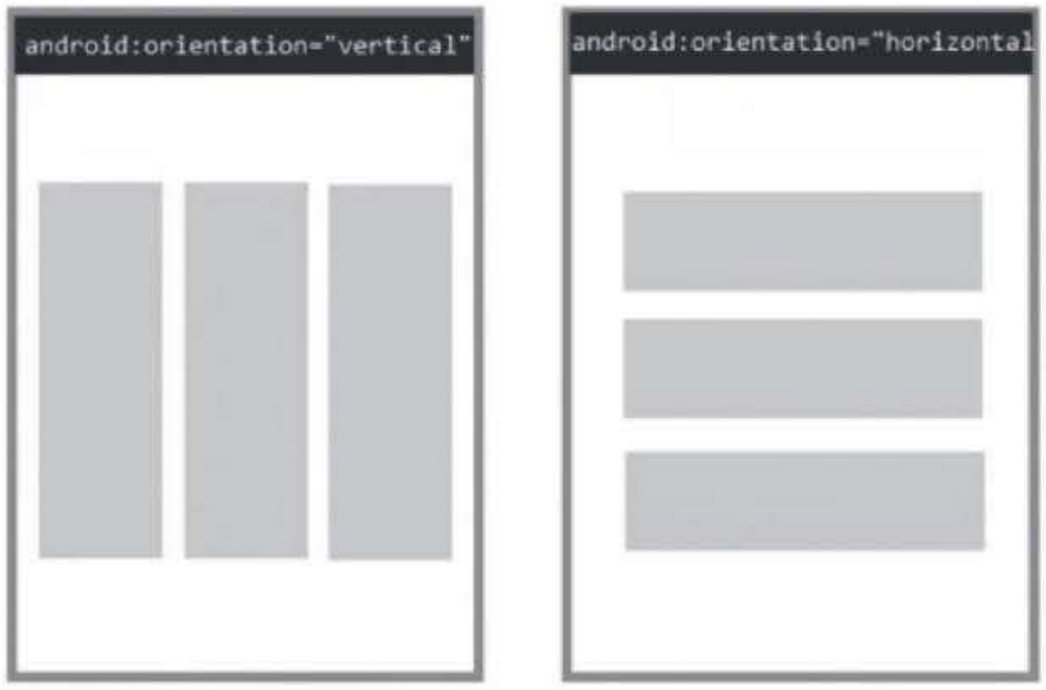
Tabla 27. Propiedades de tipo XML

PROPIEDAD	DESCRIPCIÓN
android:background	Establece un color de fondo al control View.
android:foreground	Establece un color de texto al control View.
android:id	Establece un identificador o nombre al control View.
android:minHeight	Establece una medida mínima al control View.
android:minWidth	Establece una medida máxima al control View.

android:onClick	Establece qué acción realizará el control View al presionar un clic.
android:padding	Establece un espacio interno general al control View.
android:paddingBottom	Establece un espacio desde abajo al control View.
android:paddingLeft	Establece un espacio a la izquierda del control View.
android:paddingRight	Establece un espacio a la derecha del control View.
android:paddingTop	Establece un grado de rotación del control View.
android:rotation	Establece una alineación textual del control View.
android:textAlignment	Establece la dirección textual del control View.
android:textDirection	Establece la dirección textual del control View.
android:theme	Establece un tema previamente configurado al control View.
android:tooltipText	Establece un texto informativo al presionar un control View.
android:visibility	Establece la visibilidad activa de un control View.

4.5 Control LinearLayout

Es considerado un contenedor que agrupa sus controles en una única dirección, la cual puede ser horizontal o vertical. Esta se puede especificar por medio del atributo **android:orientation**. Hay que tener en cuenta que se pueden realizar combinaciones y agrupar dentro de un linearlayout otro linearlayout, de tal forma que se podrá organizar de mejor manera los controles.



Inicialmente, el código de implementación de un LinearLayout tiene el siguiente aspecto:

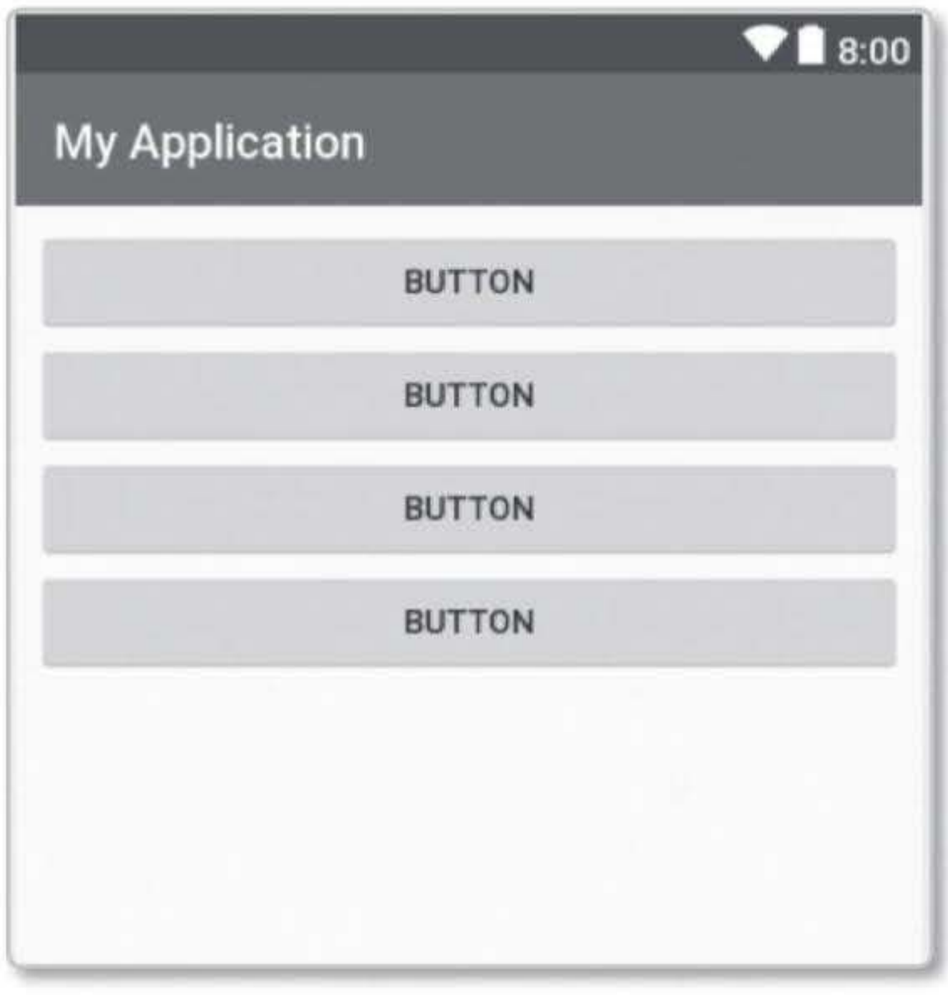
```
<LinearLayout
    android:layout_width='wrap_content'
    android:layout_height="wrap_content'
    android:orientation='vertical'
    android:id='@+id/contenedor'
    tools:layout_editor_absoluteX='8dp'
    tools:layout_editor_absoluteY='8dp'
    tools:ignore='MissingConstraints'>
</LinearLayout>
```

Si agregamos al contenedor un control de tipo Button, el código se mostraría de la siguiente manera:

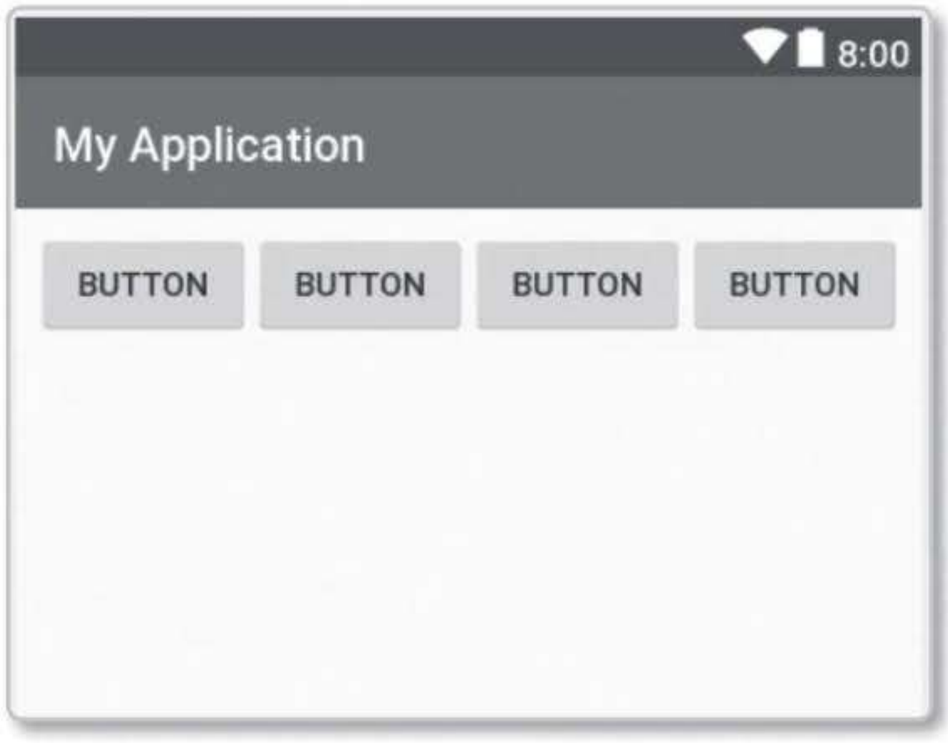
```
<LinearLayout
    android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="vertical"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">

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

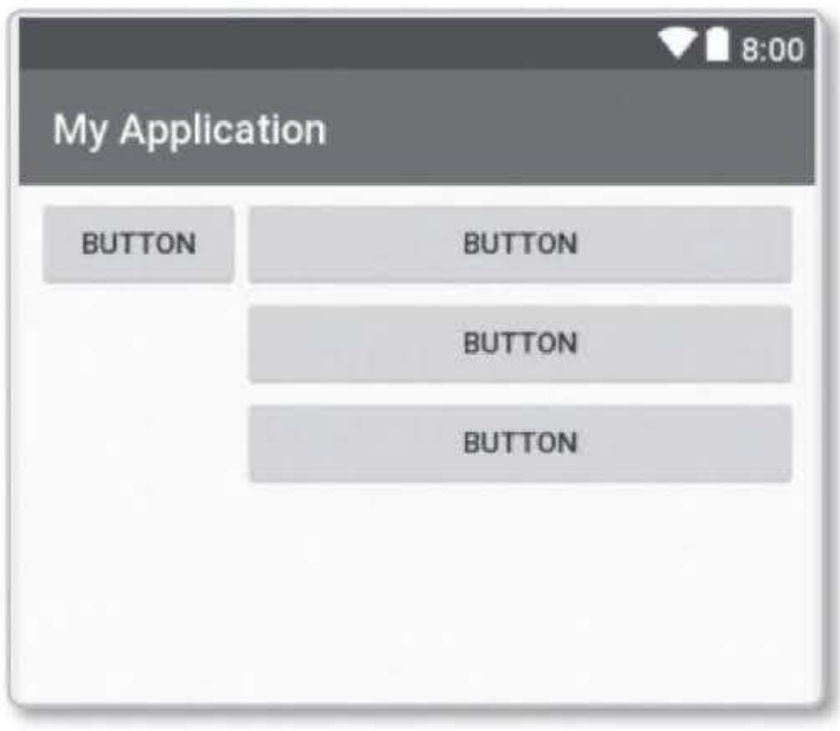
Como se observa, al agregar un control, automáticamente pasa a generar un esquema de generalización en el cual el control Button es controlado por el LinearLayout, y si agregamos más controles, estos seguirán la orientación definida en el LinearLayout, tal como se muestra en la siguiente captura, en la cual se distribuye los botones en LinearLayout con orientación vertical:



Distribución de botones en LinearLayout con orientación horizontal:



Distribución de botones integrando un LinearLayout horizontal y un vertical:

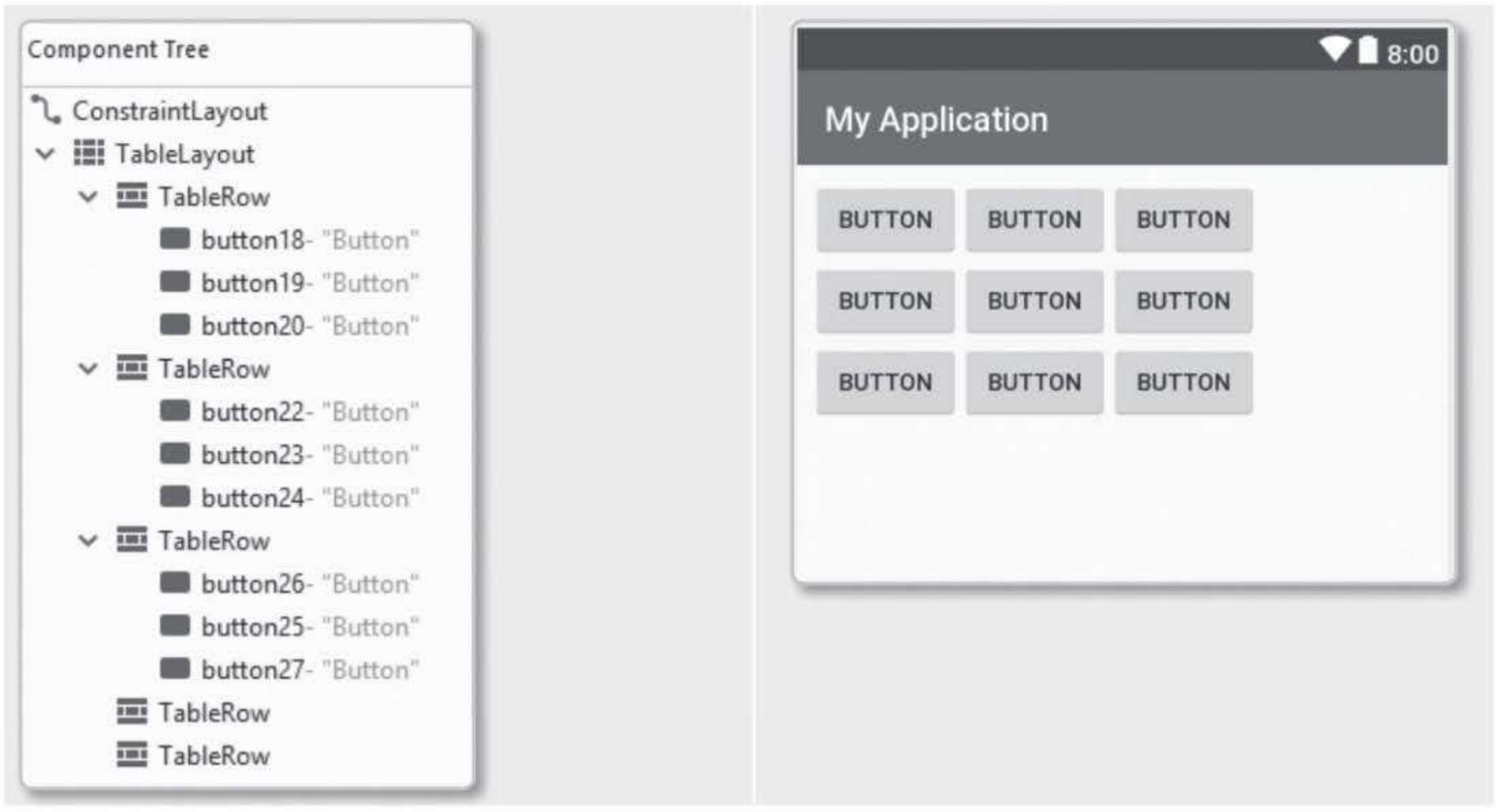


4.6 Control TableLayout

Es un contenedor matricial que define su contenido en base a filas y columnas. Cada fila del TableLayout es llamado objeto TableRow y no tiene definición de columnas. Esta se realizará al momento de agregar componentes al control TableLayout.

Con respecto al ancho de la columna, esta se controla mediante el ancho del componente agregado; es decir, el ancho de la columna lo establece el ancho del control. Asimismo, podemos observar ciertas propiedades que ayudan a la definición de los anchos:

- **android:stretchColumns**: Ajusta las celdas del lado izquierdo de acuerdo con su contenido.
- **android:shrinkColumns**: Especifica qué columnas se contraen dejando espacio en la derecha para otros controles.
- **android:collapseColumns**: Especifica qué columnas podrán ocultarse.



La jerarquía de componentes demuestra que, al agregar un componente TableLayout, automáticamente se asigna cuatro filas. Si necesita más, se debe usar el control TableRow de la paleta de controles.

Distribución de botones integrando un TableLayout.

```
<Button
    android:id="@+id/button25"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="107dp"
    android:layout_span="2"/>
```



En el script XML del último botón, observamos que se añade la propiedad **android:layout_span="2"**, la cual indica que se combinarán las dos columnas siguientes.

Distribución de botones en un *TableLayout* donde la última fila de botones combina las dos últimas columnas.

4.7 Control FrameLayout

Es un contenedor libre que permite organizar los componentes mediante propiedades (ver Figura 27). Todos los controles agregados se alinean uno encima del otro en la izquierda superior del *layout*.

- android:layout_width: Se puede especificar "match_parent" para que el ancho del *layout* se adapte al contenedor.
- android:layout_height: Se puede especificar "match_parent" para que el alto del *layout* se adapte al contenedor.
- Android:orientation: Especifica la orientación de los componentes que contendrá esta; se puede presentar de forma horizontal o vertical.

En el caso de los controles, se puede considerar las siguientes propiedades:

- Android:layout_gravity: Especifica la posición del componente dentro del FrameLayout. Este se puede presentar de las siguientes formas: arriba (top), abajo (bottom), centrado (center), centrado horizontal (center_horizontal), etc.
- Android:layout_marginLeft: Especifica el espacio que separa el lado izquierdo del contenedor y el coel componente.
- Android:scaleType: Es principalmente usado para imágenes. Aquí se especifica la forma de escala que tiene la imagen en el contenedor. Por ejemplo, para rellenar una imagen completa en el contenedor, usamos el valor "fitXY".

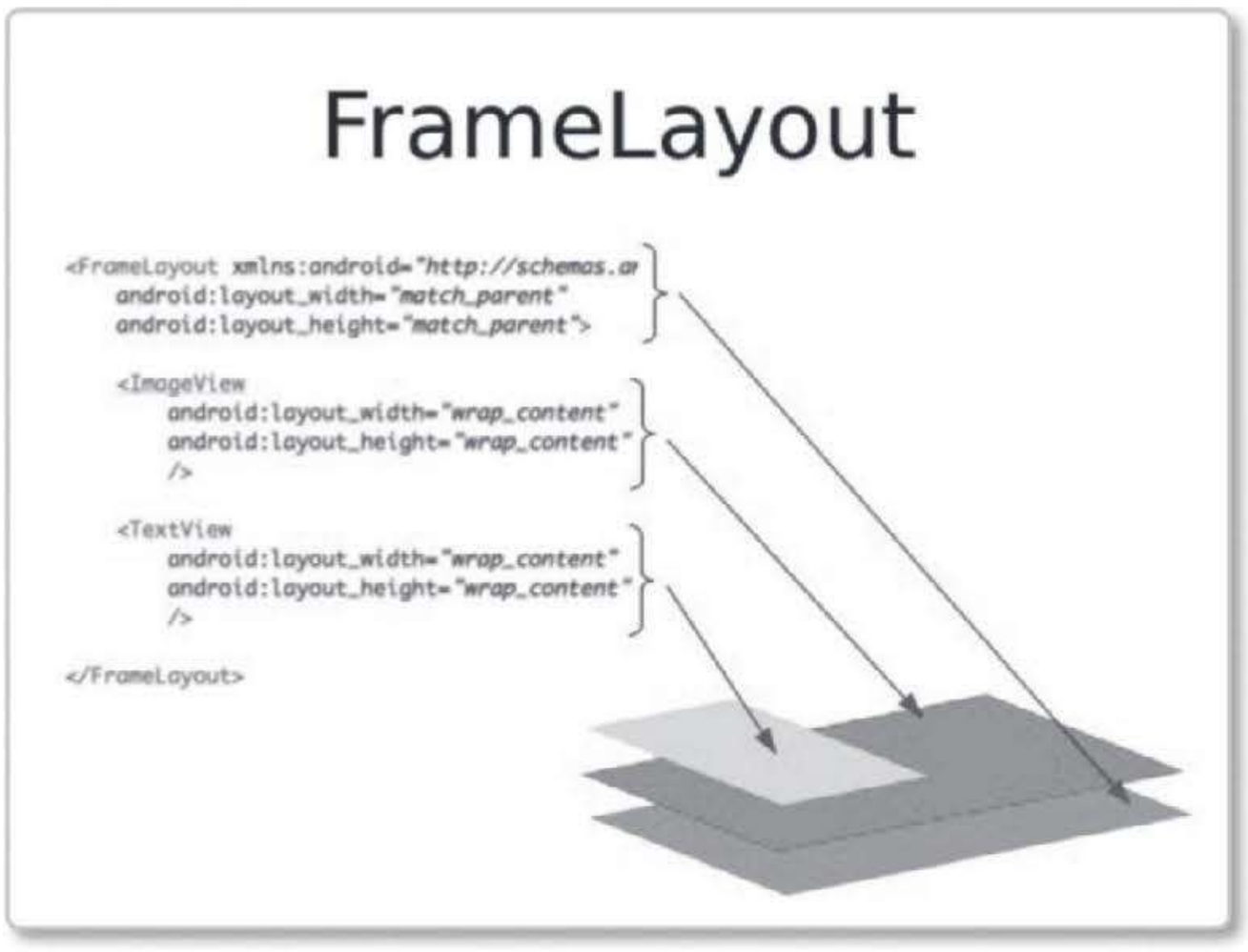


Figura 27. FrameLayout. Tomado de "Android Beginners: Views & Layouts," por A. Tripathi, 2017. Recuperado de https://medium.com/@geekanamika/android-beginners-views-layouts-657a5bbeebe2

4.8 Control TextView

Es una clase pública que permite crear objetos de tipo texto estático. Un TextView es un texto informativo en una aplicación móvil, el cual puede mostrar títulos o resultados de una operación.

Implementación de un TextView en XML:

```
<TextView
    android:id="@+id/tvTitulo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20dp"
    android:textColor="@android:color/holo_blue_bright"
    android:text="Sistema de Control de Personal" />
```

Implementación de un TextView en Java:

```
TextView tvTitulo=new TextView(getApplicationContext());
tvTitulo.setText('Sistema de Control de Personal');
tvTitulo.setTextColor(Color.BLUE);
tvTitulo.setTextSize(20);
```

En la Tabla 28, se muestran las principales propiedades del TextView en XML.

Tabla 28. Propiedades de tipo XML para TextView

PROPIEDAD	DESCRIPCIÓN
android:id	Asigna un nombre identificativo al control. Se recomienda asignar un identificador solo si el control TextView muestra un resultado de un determinado proceso. En el caso contrario, se debe mantener con su nombre inicial. Por ejemplo, podríamos asignar el nombre "tvFecha" para mostrar la fecha actual.
android:text	Asigna un valor textual que se mostrará al usuario.
android:layout_height android:layout_width	Permite asignar la dimensión que tiene el control TextView con respecto al *layout*. De forma estándar, aparece el valor **wrap content**, el cual indica que las dimensiones del control se ajustarán al contenido.
android:gravity	Permite asignar una alineación del texto dentro del marco del control.
android:textSize	Permite modificar el tamaño del texto.
android:background	Permite asignar un color de fondo al control TextView.
android:textColor	Permite asignar un color al texto del control TextView.
android:typeface	Permite asignar un tipo de letra al texto contenido en el control TextView.

4.9 Control EditText

Tiene las mismas características que un control TextView, cuya apariencia ha sido modificada para que tenga un comportamiento de campo de texto. Es decir, el usuario podrá interactuar con la aplicación por medio de ingreso de información. A continuación, veamos una lista de tipos de entrada que se puede configurar para un control EditText. Esta referencia permite cambiar el teclado al momento de ingresar un determinado valor (ver Tabla 29).

Tabla 29. Tipos de entrada para EditText

PROPIEDAD	DESCRIPCIÓN
Plain Text	Proporciona características de ingreso de cualquier tipo de carácter.
Person Name	Proporciona características de ingreso de solo textos.
Password (numeric)	Proporciona características de ingreso de números enmascarados.
E-mail	Proporciona características de ingreso de correo electrónico.
Phone	Proporciona características de ingreso de números telefónicos.
Postal Address	Proporciona características de ingreso de direcciones postales.
Multiline Text	Proporciona características de ingreso de textos en varias líneas.
TimE	Proporciona características de ingreso de formatos tipo hora.
Date	Proporciona características de ingreso de formatos tipo fecha.
NumBer	Proporciona características de ingreso de tipo numérico.
Number (signed)	Proporciona características de ingreso de tipo numérico sin signo.
Number (decimal)	Proporciona características de ingreso de tipo numérico real.
Password	Proporciona características de ingreso de caracteres enmascarados.

Implementación de un EditText en XML:

```
<EditText
    android:id='@+id/etNombre'
    android:layout_width='match_parent'
    android:layout_height='wrap_content'
    android:inputType='text'
    android:hint='Ingrese Nombre' />
```

Implementación de un EditText en Java:

```
EditText etNombre=new EditText(getApplicationContext());
etNombre.setHint('Ingrese Nombre:');
etNombre.setInputType(1);
```

En la Tabla 30, se muestran las principales propiedades del EditText en XML.

Tabla 30. Propiedades de tipo XML para EditText

PROPIEDAD	DESCRIPCIÓN
android:id	Asigna un nombre identificativo al control. Por ejemplo, para el ingreso del nombre del usuario, le podríamos llamar etUsuario.
android:text	No es recomendado llenar esta propiedad, pues es el usuario quien ingresara algún valor al control cuando se encuentre en el tiempo de ejecución de la aplicación.
android:hint	Asigna un texto descriptivo al control EditText, el cual permite al usuario visualizar información referente a cómo debe llenar correctamente el control.
android:inputType	Configura el tipo de valor que se ingresará por el control EditText. Esto permite al usuario ver un teclado en su dispositivo de acuerdo con el valor que ingresará.
android:maxLength	Define la cantidad de caracteres como límite que puede tener el control EditText.
android:singleLine	Permite ingresar un texto en una sola línea, bloqueando el botón de cambio de línea.
android:enabled	Determina si son o no editables los datos mostrados en un EditText.

4.10 Control Button

Es un control de tipo acción que permite dar respuesta a un determinado evento. Este control incluye un texto o una imagen.

Implementación de un *Button* en XML:

```
<Button
    android:id='@+id/btProcesar'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:text='PROCESAR' />
```

Implementación de un *Button* en Java:

```
Button btProcesar=new Button(getApplicationContext());
btProcesar.setText('PROCESAR');
```

En la Tabla 31, se aprecian las principales propiedades en XML.

Tabla 31. Propiedades de tipo XML para Button

ATRIBUTO	DESCRIPCIÓN
android:text	Asigna un texto descriptivo al botón.
android:background	Asigna un color de fondo al botón. Hay que tener en cuenta que este se puede dar por medio de los recursos de la aplicación o directamente por código hexadecimal.
android:enabled	Permite habilitar el botón para su uso por defecto. Tiene el valor True.
android:gravity	Permite asignar una ubicación del texto sobre los ejes del botón.
android:id	Permite asignar un nombre único dentro de la aplicación al botón.
android:onClick	Asocia un método implementado en Java al botón para que este realice una determinada actividad.
android:textColor	Asigna un color al texto contenido del botón.

Asimismo, para asignar un evento al botón, se puede adoptar las siguientes maneras:

Primera forma. Creamos un método en Java y se asigna OnClick en XML.

- Creamos un método llamado mostrarMensaje en el archivo **main_activity.java**.

```
public void mostrarMensaje(View view){
    Toast.makeText(getApplicationContext(),
                   'Mensaje desde un boton',Toast.LENGTH_SHORT).show();
}
```

- Agregamos un botón desde el archivo **activity_main.xml**.

```
<Button
    android:id='@+id/btMensaje'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:onClick='mostrarMensaje'
    android:text='MOSTRAR MENSAJE' />
```

Segunda forma. Creamos el método onClick dentro del método OnCreate en Java.

- Agregamos un botón desde el archivo **activity_main.xml**.

```
<Button
    android:id='@+id/btMensaje'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:text='MOSTRAR MENSAJE' />
```

- Agregamos el método setOnClickListener del botón en el archivo **main_activity.java**.

```
Button btMensaje;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
   btMensaje = (Button) findViewById(R.id.btMensaje);
    btMensaje.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View v) {
            Toast.makeText(getApplicationContext(),
                    'Mensaje desde un boton',Toast.LENGTH_SHORT).show();
        }
    });
}
```

Tercera forma. Creamos el método OnClickListener independiente en Java.

- Agregamos un botón desde el archivo **activity_main.xml**

```
<Button
    android:id='@+id/btMensaje'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:text='MOSTRAR MENSAJE' />
```

- Creamos un método llamado mostrarMensaje del tipo OnClickListener en el archivo **main_activity.java**.

```
Button btMensaje;

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

    btMensaje = (Button) findViewById(R.id.btMensaje);
    btMensaje.setOnClickListener(mostrarMensaje);
}

private View.OnClickListener mostrarMensaje = new View.OnClickListener() {
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(),
                'Mensaje desde un boton',Toast.LENGTH_SHORT).show();
    }
};
```

Cuarta forma. Creamos un método genérico OnClick en Java.

- Agregamos un botón desde el archivo **activity_main.xml**.

```
Button
    android:id='@+id/btMensaje'
    android:layout_width='wrap_content'
    android:layout_height='wrap_content'
    android:text='MOSTRAR MENSAJE' />
```

- Creamos un método llamado OnClick en el archivo **main_activity.java**.

```
public class MainActivity extends AppCompatActivity
                                    implements View.OnClickListener {

    //Declarar objeto de la clase Button
    Button btMensaje;

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

        //Asociar el boton al control
        btMensaje = (Button) findViewById(R.id.btMensaje);
        btMensaje.setOnClickListener(this);
    }

    //Definicion del metodo OnClick
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(),
                'Mensaje desde un boton',Toast.LENGTH_SHORT).show();
    }
}
```

4.11 Control ScrollView

Es considerado un tipo especial de FrameLayout, ya que permite desplazarse de forma horizontal o vertical por la pantalla cuando los componentes excedan el espacio móvil. Al crear un ScrollView, automáticamente se agrega un LinearLayout para alinear sus componentes.



Implementación de un Button en XML:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"
>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" />
</ScrollView>
```

Implementación de un Button en Java:

```
ScrollView svImagenes=new ScrollView(getApplicationContext());
```

En la Tabla 32, se aprecia las principales propiedades en XML.

Tabla 32. Propiedades de tipo XML para ScrollView

ATRIBUTO	DESCRIPCIÓN
android:layout_width	Asigna el ancho que ocupará el ScrollView.
android:layout_height	Asigna el alto que ocupará el ScrollView.
android:fillViewport	Activa que el ScrollView tenga un soporte en todos los dispositivos. Es también llamado aplicación móvil responsive.
android:scrollbars	Especifica qué tipo de barra de desplazamiento se mostrará en el Scroll.

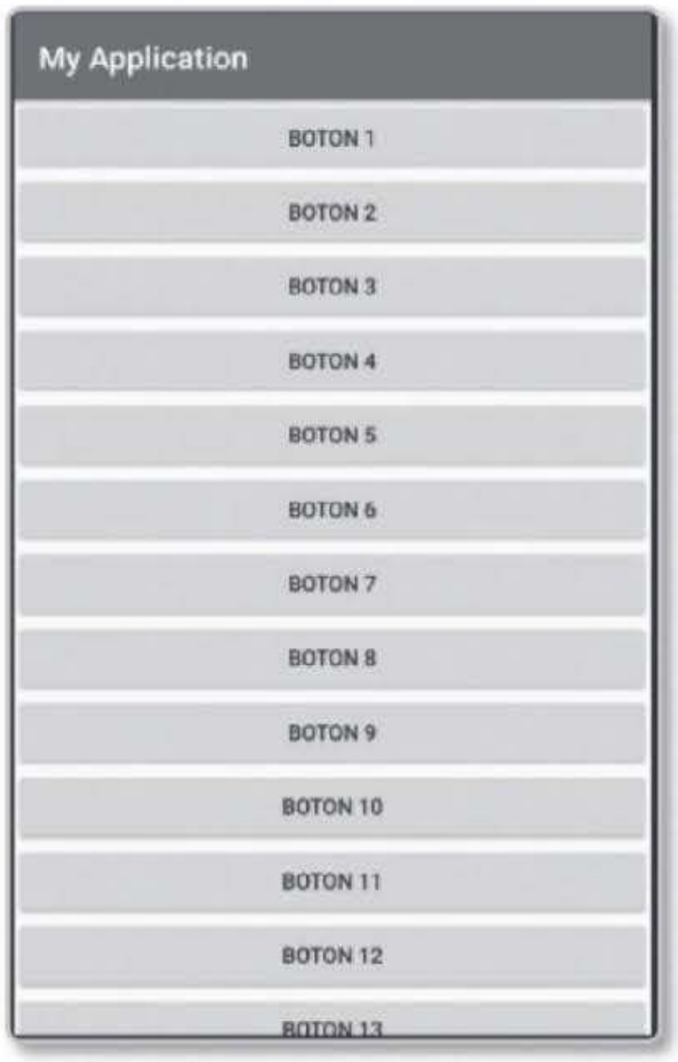
Por ejemplo, usando Java, podemos crear 20 botones dentro de un ScrollView mediante el siguiente código:

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

    LinearLayout contenedor = new LinearLayout(this);
    contenedor.setOrientation(LinearLayout.VERTICAL);
    contenedor.setLayoutParams(new LinearLayout
                    .LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
                      LinearLayout.LayoutParams.MATCH_PARENT));

    for (int i = 1; i <= 20; i++)
    {
        Button bt = new Button (this);
        bt.setText("Boton "+i);
        contenedor.addView(bt, new LinearLayout.LayoutParams(LinearLayout
                .LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.MATCH_
PARENT));
    }

    ScrollView sv = new ScrollView(this);
    sv.setLayoutParams(new LinearLayout
                    .LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
                    LinearLayout.LayoutParams.MATCH_PARENT));
   sv.setVerticalScrollBarEnabled(true);
    sv.addView(contenedor);
    setContentView(sv);
}
```



4.12 **Control ImageView**

Permite mostrar una imagen en una aplicación Android. Esta debe encontrase previamente almacenada en el proyecto (ver Figura 28).



Figura 28. Modelo de ImageView. Tomado de "1- What is ImageView?," por o7planning, s.f. Recuperado de https://o7planning.org/en/10531/android-imageview-tutorial

Implementación de un ImageView en XML:

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/imagen" />
```

Implementación de un ImageView en Java:

```
ImageView ivImagen = new ImageView(getApplicationContext());
ivImagen.setImageResource(R.drawable.miImagen);
```

En la Tabla 33, se expone las principales propiedades en XML.

Tabla 33. Propiedades de tipo XML para ImageView

ATRIBUTO	DESCRIPCIÓN
android:layout_width	Asigna el ancho que ocupará el ImageView sobre su contenedor.
android:layout_height	Asigna el alto que ocupará el Imageview sobre su contenedor.
android:src	Define la imagen que se mostrará. Esta deberá previamente estar aso-ciada con el proyecto.
android:id	Asigna un nombre al control ImageView.
android:scaleType	Asigna una escala relativa a su contenedor. Se recomienda usar fitXY para ajustar completamente la imagen.

4.13 Control ImageButton

Permite usar una imagen como botón. Esto permitirá accionar eventos por medio de imágenes y dejará de lado solo la exposición de una imagen.

Implementación de un ImageView en XML:

```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/portada" />
```

Implementación de un ImageView en Java:

```
ImageButton ibImagen = new ImageButton(getApplicationContext());
ibImagen.setImageResource(R.drawable.miImagen);
```

En la Tabla 34, se muestran las principales propiedades en XML.

Tabla 34. Propiedades de tipo XML para ImageButton

ATRIBUTO	DESCRIPCIÓN
android:src	Asigna una imagen al botón.

android:enabled	Permite habilitar el botón para su uso por defecto; tiene el valor True.
android:id	Permite asignar un nombre único dentro de la aplicación al botón.
android:onClick	Asocia un método implementado en Java al botón para que este realice una determinada actividad.
android:scaleType	Asigna una escala relativa. Se recomienda usar fitXY para ajustar la imagen sobre el botón.

4.14 Control Spinner

Muestra una lista desplegable tipo cuadro combinado, desde la cual se podrá seleccionar un único elemento, tal como se expone en la Figura 29.

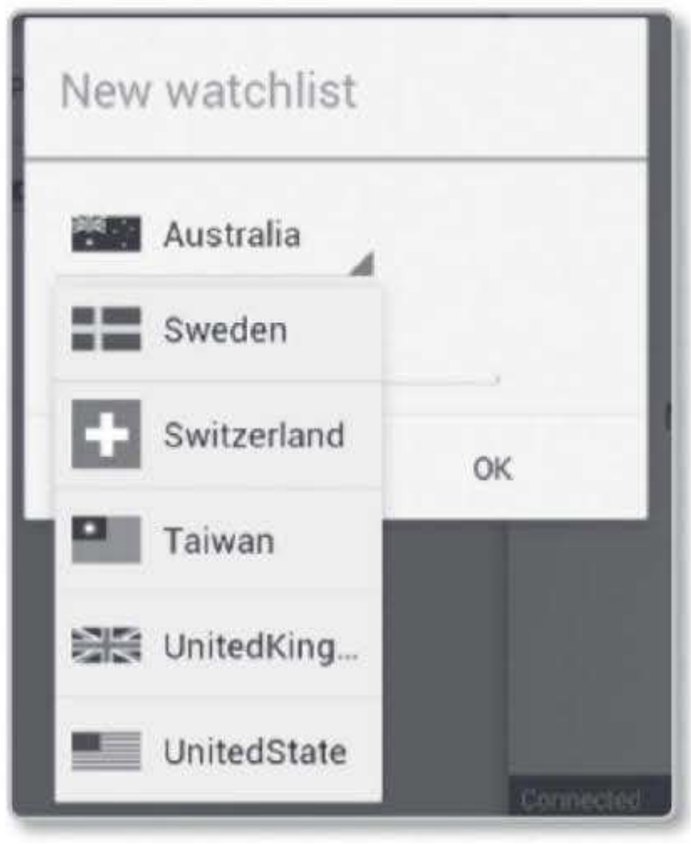


Figura 29. Modelo Spinner. Tomado de "How can I make the spinner width same as largest available item width," por Stackoverflow, 2013. Recuperado de https://stackoverflow.com/questions/16633705/how-can-i-make-the-spinner-width-same-as-largest-available-item-width

Es necesario declarar los valores que se mostrarán en el control Spinner. Este código se debe colocar dentro del archivo Strings.xml.

```
<resources>
    <string-array name='dias'>
        <item>Lunes</item>
        <item>Martes</item>
        <item>Miercoles</item>
        <item>Jueves</item>
        <item>Viernes</item>
        <item>Sabado</item>
        <item>Domingo</item>
    </string-array>
</resources>
```

Implementación del Spinner en Java:

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

En la Tabla 35, se muestran las principales propiedades en XML.

Tabla 35. Propiedades de tipo XML para Spinner

ATRIBUTO	DESCRIPCIÓN
android:layout_width	Asigna el ancho que ocupará el Spinner sobre su contenedor.
an-droid:layout_height	Asigna el alto que ocupará el Spinner sobre su contenedor.
android:id	Permite asignar un nombre único dentro de la aplicación.
android:entries	Permite asociar un conjunto de elementos con el Spinner.

4.15 Control CheckBox

Es un control que permite marcar y desmarcar opciones de una aplicación.



Implementación de un ImageView en XML:

```
<CheckBox
    android:id='@+id/checkBox'
    android:layout_width='match_parent'
    android:layout_height='wrap_content'
    android:text='CheckBox' />
```

Implementación de un ImageView en Java:

```
CheckBox cbOpcion =new CheckBox(getApplicationContext());
cbOpcion.setText('Opcion 1');
```

En la Tabla 36, se exponen las principales propiedades en XML.

Tabla 36. Propiedades de tipo XML para CheckBox

ATRIBUTO	DESCRIPCIÓN
android:layout_width	Especifica el ancho del control sobre su contenedor.
an-droid:layout_height	Especifica el alto del control sobre su contenedor.
android:text	Especifica el texto que se mostrará al usuario.
android:checked	Especifica si el control inicia seleccionado o no.

4.16 Control ProgressBar

Es un control que permite indicar el progreso de una determinada tarea (ver Figura 30).

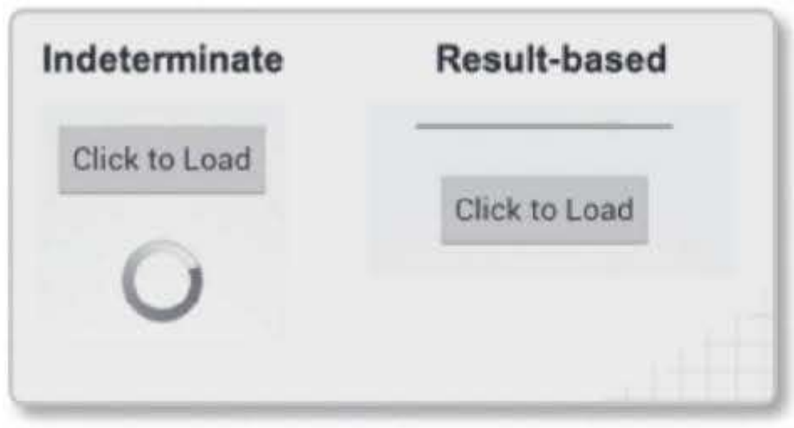


Figura 30. Modelo ProgressBar. Tomado de "Handling ProgressBars, " por Codepath, s.f. Recuperado de https://guides.codepath.com/android/handling-progressbars.

Implementación de un ImageView en XML:

```
<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyle"
    android:progress="10"
    android:max="10"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Implementación de un ImageView en Java:

```
ProgressBar pbProgreso =new ProgressBar(getApplicationContext());
contenedor.addView(pbProgreso);
```

En la Tabla 37, se aprecian las principales propiedades en XML.

Tabla 37. Propiedades de tipo XML para ProgressBar

ATRIBUTO	DESCRIPCIÓN
android:layout_width	Especifica el ancho del control sobre su contenedor.
an-droid:layout_height	Especifica el alto del control sobre su contenedor.
android:max	Controla el nivel de porcentaje máximo.
android:id	Asigna un nombre al control.
android:progress	Muestre el progreso en porcentaje.

4.17 **Dialog Window**

Es una ventana pequeña que espera la acción del usuario para realizar una determinada acción, la cual puede ser seleccionar alguna opción o ingresar algún valor. Normalmente, esta ventana no ocupa toda la pantalla y exige la selección de una opción para continuar con el proceso (ver Figura 31).



Figura 31. Modelo de Dialog Window. Tomado de "How can I change default black dim background "color" (not the amount of dim) of Dialog?," por Stackoverflow, 2015. Recuperado de https://stackoverflow.com/questions/29227126/how-can-i-change-default-black-dim-background-color-not-the-amount-of-dim-of/29482234

Por ejemplo, para mostrar un cuadro de diálogo que tenga las opciones Sí y No, y que al seleccionar muestre un mensaje de la selección realizada, se realiza lo siguiente:

Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:id="@+id/btMostrar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="MOSTRAR DIALOGO" />
```

Implementación del AlertDialog en MainActivity.java:

```
btMostrar=(Button)findViewById(R.id.btMostrar);
btMostrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setTitle("Dialogo");

        builder.setMessage("Esta seguro de salir?");
```

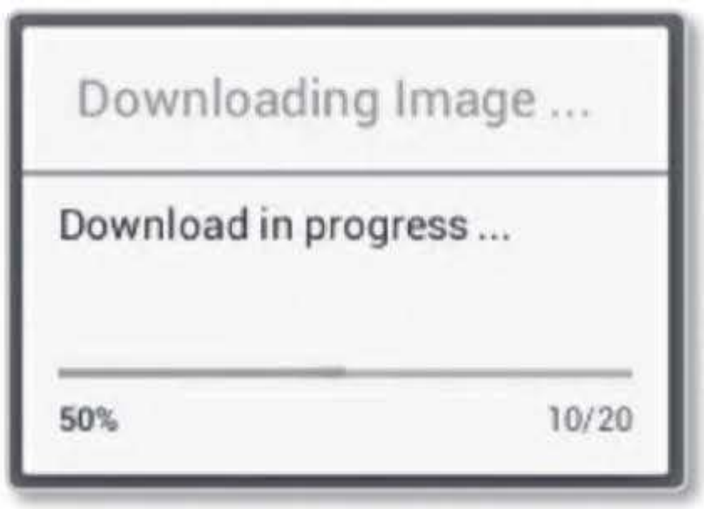
```
        builder.setPositiveButton('Yes', new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(getApplicationContext(),
                        'Ha seleccionado SI',Toast.LENGTH_SHORT).show();
            }
        });

        builder.setNegativeButton('No', new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(getApplicationContext(),
                        'Ha seleccionado NO',Toast.LENGTH_SHORT).show();
            }
        });

        AlertDialog dialog - builder.create();
        dialog.show();
    }
});
```

4.18 Progress Dialog

Es un control que permite mostrar una barra de progreso asociado con un proceso dentro de la aplicación móvil. Aquél es especializado en hacer esperar a un usuario mientras se desarrolla algo.



Por ejemplo, mostrar una barra de progreso simulando un envío de valores se realiza del siguiente modo.

Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:text='Mostrar Dialogo de Progreso'
    android:layout_width='match_parent'
    android:layout_height='wrap_content'
    style='@android:style/Widget.DeviceDefault.Button.Inset'
    android:textSize='15dp'
    android:id='@+id/btMostrar'
    android:textColor='@android:color/background_dark' />
```

Implementación del Progress en **MainActivity.java**:

```
Button btMostrar;
ProgressDialog progressDialog;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
        Handler handle = new Handler() {
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            progressDialog.incrementProgressBy(2);
        }
    };

    @Override
    public void onClick(View v) {
        progressDialog = new ProgressDialog(MainActivity.this);
        //Definir el maximo valor del progreso
        progressDialog.setMax(100);
        //Especificar el mensaje al usuario
        progressDialog.setMessage('Enviando...');
        //Especificar el titulo del progreso
        progressDialog.setTitle('ProgressDialog');
        //Definicion de progreso horizontal
       progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        //Mostrar el dialogo de progreso
        progressDialog.show();
        progressDialog.setCancelable(false);
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
             while (progressDialog.getProgress() <= progressDialog.getMax()) {
                        Thread.sleep(200);
                        handle.sendMessage(handle.obtainMessage());
                if (progressDialog.getProgress() == progressDialog.getMax()) {
                            progressDialog.dismiss();
                        }
                    }
                } catch (Exception e) {
                }
            }
        }).start();
    }
});
```

4.19 **Diálogos personalizados**

Un diálogo en Android es considerado un elemento que se superpone al activity, el cual se puede emplear para agregar algún valor o responder según lo crea necesario la aplicación. Una de sus características es que interrumpe una actividad y no desaparece hasta que el usuario reaccione ante el diálogo.

Diálogo simple. Presenta las características que se muestran a continuación.



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:text="Mostrar Dialogo Simple"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@android:style/Widget.DeviceDefault.Button.Inset"
    android:textSize="15dp"
    android:id="@+id/btMostrar"
    android:textColor="@android:color/background_dark" />
```

Implementación del diálogo en MainActivity.java:

```
Button btMostrar;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        AlertDialog.Builder builder=new AlertDialog.Builder(MainActivity.this);
            builder.setIcon(R.mipmap.ic_launcher);
            builder.setTitle("Titulo del Dialogo");
            builder.setMessage("Mensaje desde el Dialogo Simple");
            AlertDialog alertDialog = builder.create();
            alertDialog.show();
        }
    });
}
```

Diálogo con lista de opciones simple. Presenta las características que se exponen a continuación.



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:text="Mostrar Dialogo de Alerta"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@android:style/Widget.DeviceDefault.Button.Inset"
    android:textSize="15dp"
    android:id="@+id/btMostrar"
    android:textColor="@android:color/background_dark" />
```

Implementación del diálogo en MainActivity.java:

```
Button btMostrar;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        AlertDialog.Builder builder=new AlertDialog.Builder(MainActivity.this);
            final CharSequence[] items = new CharSequence[3];

            items[0] = 'Opción 1';
            items[1] = 'Opción 2';
            items[2] = 'Opción 3';

            builder.setTitle('Diálogo De Lista')
                    .setItems(items, new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int i) {
                            Toast.makeText(
                                    MainActivity.this,
                                    'Usted ha seleccionado:' + items[i],
                                    Toast.LENGTH_SHORT).show();
                        }
                    });

            AlertDialog alertDialog = builder.create();
            alertDialog.show();
        }
    });
}
```

Diálogo con lista de opciones. Presenta las características que se aprecian a continuación.



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:text="Mostrar Dialogo de Alerta"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@android:style/Widget.DeviceDefault.Button.Inset"
    android:textSize="15dp"
    android:id="@+id/btMostrar"
    android:textColor="@android:color/background_dark" />
```

Implementación del diálogo en MainActivity.java:

```
Button btMostrar;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        AlertDialog.Builder builder=new AlertDialog.Builder(MainActivity.this);
            final CharSequence[] items = new CharSequence[3];

            items[0] = 'Opción 1';
            items[1] = 'Opción 2';
            items[2] = 'Opción 3';

        builder.setTitle('Seleccione una opción')
        .setSingleChoiceItems(items, 0, new DialogInterface.OnClickListener() {
                    @Override
                public void onClick(DialogInterface dialog, int i) {
                    Toast.makeText(
                            MainActivity.this,
                            'Seleccionaste: ' + items[i],
                            Toast.LENGTH_SHORT).show();
                }
            });
        AlertDialog alertDialog = builder.create();
        alertDialog.show();
        }
    });
}
```

Diálogo con lista de opciones en Checks. Presenta las características que se muestran a continuación.



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<Button
    android:text="Mostrar Dialogo de Alerta"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@android:style/Widget.DeviceDefault.Button.Inset"
    android:textSize="15dp"
    android:id="@+id/btMostrar"
    android:textColor="@android:color/background_dark" />
```

Implementación del diálogo en MainActivity.java:

```
Button btMostrar;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    AlertDialog.Builder builder=new AlertDialog.Builder(MainActivity.this);

        final ArrayList itemsSeleccionados = new ArrayList();
        final CharSequence[] items = new CharSequence[3];

        items[0] = 'Opción 1';
        items[1] = 'Opción 2';
        items[2] = 'Opción 3';

        builder.setTitle('Intereses').setMultiChoiceItems(items, null,
                    new DialogInterface.OnMultiChoiceClickListener() {
```

```
                @Override
                public void onClick(DialogInterface dialog, int i,
                                    boolean isChecked) {
                    if (isChecked) {
                        itemsSeleccionados.add(i);
                        Toast.makeText(MainActivity.this,
                                "Checks seleccionados:(' +
                                    itemsSeleccionados.size() + ')',
                                Toast.LENGTH_SHORT).show();
                    } else if (itemsSeleccionados.contains(i)) {
                itemsSeleccionados.remove(Integer.valueOf(which));
                                }
                            }
                        });
                    AlertDialog alertDialog = builder.create();
                    alertDialog.show();
                }
            });
    }
```

Diálogo con AlertDialog. Veamos cómo invocar a un AlertDialog a partir de un botón. Al seleccionar SÍ, la aplicación se cerrará; en el caso contrario, solo cerrará el diálogo y seguirá dentro de la aplicación.



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <Button
        android:id='@+id/btSalir'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:layout_weight='1'
        android:text='SALIR' />
</LinearLayout>
```

Implementación del diálogo en MainActivity.java:

```
package com.torres.manuel.myapplication;

import android.app.Dialog;
import android.content.DialogInterface;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

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

        Button btSalir = (Button)findViewById(R.id.btSalir);
        btSalir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
          AlertDialog.Builder dialogo=new AlertDialog.Builder(MainActivity.this);
                dialogo.setMessage('¿Salir?');
               dialogo.setPositiveButton('Si', new DialogInterface.
OnClickListener() {
                    @Override
                   public void onClick(DialogInterface dialog, int which) {
                        MainActivity.this.finish();
                    }
                });
               dialogo.setNegativeButton('No', new DialogInterface.
OnClickListener() {
                    @Override
                   public void onClick(DialogInterface dialog, int which) {
                        dialog.cancel();
                    }
                });
                dialogo.show();
            }
        });
    }
}
```

Diálogo con ProgressDialog. Veamos cómo invocar a un ProgressDialog a partir de un botón. Este muestra un círculo de progreso a partir del mensaje Descargando, y espera mientras descarga, tal como se muestra en la siguiente captura:



Implementación de un botón para mostrar el cuadro de diálogo en XML:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <Button
        android:id='@+id/btSalir'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:layout_weight='1'
        android:text='SALIR' />
</LinearLayout>
```

Implementación del diálogo en **MainActivity.java**:

```
package com.torres.manuel.myapplication;

import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;

public class MainActivity extends AppCompatActivity {

    ProgressDialog dialogo;

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

        Button btSalir = (Button)findViewById(R.id.btSalir);
        btSalir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                dialogo = new ProgressDialog(MainActivity.this);
                dialogo.setMax(100);
                dialogo.setMessage('Espera mientras descarga');
                dialogo.setTitle('Descargando');
                dialogo.show();
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            while(dialogo.getProgress()<=dialogo.getMax()){
                                Thread.sleep(200);
                                handler.sendMessage(handler.obtainMessage());
                                if(dialogo.getProgress() == dialogo.getMax()){
                                    dialogo.dismiss();
```

```
                                }
                            }
                        }catch (Exception ex){
                            ex.printStackTrace();
                        }
                    }
                }).start();
            }
        });
    }
    Handler handler=new Handler(){
        @Override
        public void handleMessage(Message msg){
            super.handleMessage(msg);
            dialogo.incrementProgressBy(1);
        }
    };
}
```

4.20 Notificaciones con Toast

Las notificaciones Toast proporcionan información corta sobre una determinada acción, normalmente en la parte inferior del dispositivo. Esta desaparece automáticamente después de un tiempo.



Notificación Toast a partir de un objeto:

```
Toast mensaje = Toast.makeText(getApplicationContext(),
                'Mensaje', Toast.LENGTH_SHORT);
mensaje.show();
```

Notificación Toast sin objeto:

```
Toast.makeText(getApplicationContext(),
                'Mensaje', Toast.LENGTH_SHORT).show();
```

Notificación Toast en el centro del aplicativo:

```
Toast mensaje = Toast.makeText(getApplicationContext(),
                'Mensaje', Toast.LENGTH_SHORT);
mensaje.setGravity(Gravity.CENTER|Gravity.CENTER,0,0);
mensaje.show();
```

Personalización del mensaje Toast:



Creación de un diseño en un nuevo **Activity Main** llamado **activity_mi_mensaje.xml**. Este diseño será el que se muestre en la notificación Toast.

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    android:id='@+id/miContenedor'
    android:orientation='horizontal'
    android:layout_width='fill_parent'
    android:layout_height='fill_parent'
    android:padding='8dp'
    android:background='#DAAA'
    >
    <ImageView android:src='@mipmap/ic_launcher'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:layout_marginRight='8dp'
        />
    <TextView android:id='@+id/tvMensaje'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:textColor='#FFF'
        />
</LinearLayout>
```

Creación de un botón para mostrar la notificación en el archivo **activity_main.xml**.

```
LinearLayout
    android:id='@+id/contenedor'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'>

    <Button
        android:text='Mostrar Notificación Toast'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        style='@android:style/Widget.DeviceDefault.Button.Inset'
        android:textSize='15dp'
        android:id='@+id/btMostrar'
        android:textColor='@android:color/background_dark' />
</LinearLayout>
```

Implementación en el MainActivity.java:

```
Button btMostrar;

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

    btMostrar=(Button)findViewById(R.id.btMostrar);
    btMostrar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            LayoutInflater inflater = getLayoutInflater();
            View layout = inflater.inflate(R.layout.activity_mi_mensaje,
                    (ViewGroup) findViewById(R.id.miContenedor));

           TextView text = (TextView) layout.findViewById(R.id.tvMensaje);
            text.setText('Notificación Personalizada con Toast');

            Toast toast = new Toast(getApplicationContext());
            toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
            toast.setDuration(Toast.LENGTH_LONG);
            toast.setView(layout);
            toast.show();
        }
    });
}
```

4.21 Menú de opciones

Los menús son parte importante de nuestras aplicaciones e integrar uno nuevo a nuestro proyecto es relativamente fácil: solo es cuestión de modificar dos archivos. En Android, existen tres tipos de menús. Estos se exponen a continuación.

- **Menú de opciones:** Se encuentra implementado en un activity de tipo Menu resource file, que luego será asociado con otro activity que mostrará el menú.



- **Menú contextual:** Se caracteriza por ser flotante y aparece cuando seleccionamos prolongadamente un botón.

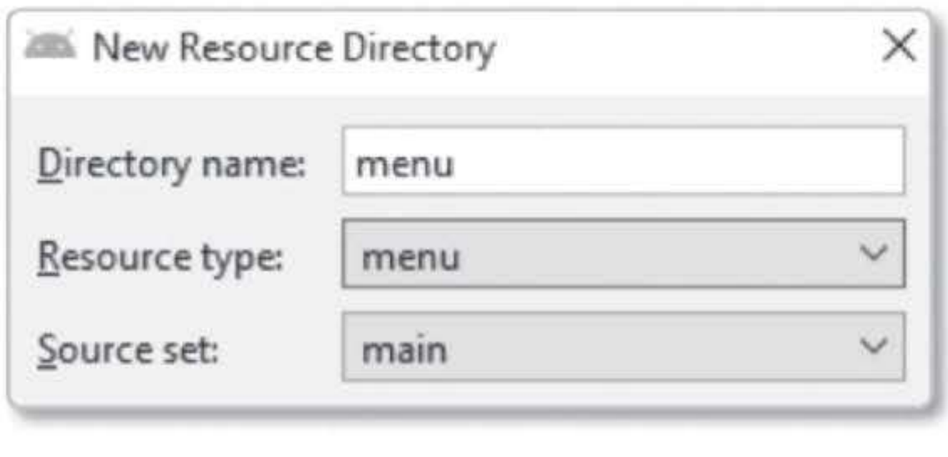


- **Menú *pop up*:** Permite mostrar un menú contextual en una ventana pop up, la cual se ubica junto al elemento que lo invoca.

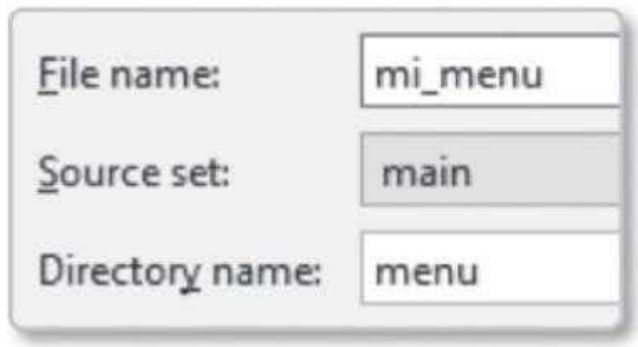


Por ejemplo, para desarrollar un menú de opciones, podemos seguir los siguiente pasos:

1. Creamos un contenedor de menú presionando clic derecho en el **proyecto > new > Android Resource Directory >**.
2. Asignamos las siguientes opciones:



3. Notaremos que se ha creado una carpeta **Menú** dentro del paquete **res** de la aplicación. Aquí presionamos clic derecho **> new > Menú resource file,** y asignamos un nombre.



4. Colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<menu xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'>
    <item android:id='@+id/opcion1'
        android:title='Opcion 1'
        app:showAsAction='never'
        />
    <item android:id='@+id/opcion2'
        android:title='Opcion 2'
        app:showAsAction='never'
        />
</menu>
```

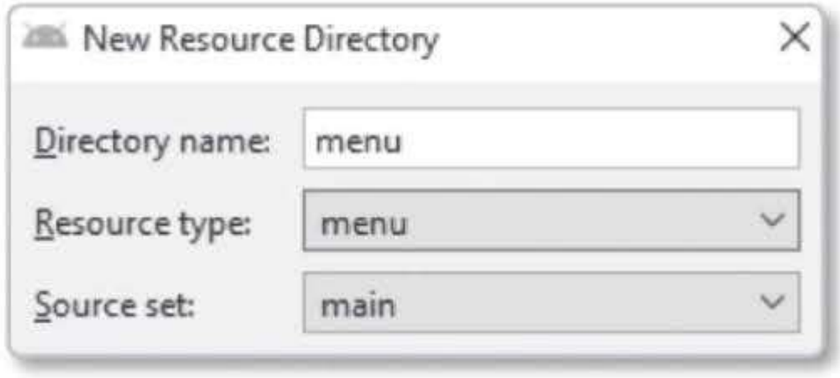
5. Invocamos el menú en el activity que lo mostrará usando el siguiente código:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()){
        case R.id.menu1:
            Toast.makeText(this,'Selecciono MENU',Toast.LENGTH_SHORT).
show();
            return true;
        case R.id.opcion1:
            Toast.makeText(this,'Selecciono la subopcion 1',
                                                Toast.LENGTH_SHORT).show();
            return true;
            case R.id.opcion2:
            Toast.makeText(this,'Selecciono la subopcion 2',
                                                Toast.LENGTH_SHORT).show();
                return true;
    }
    return super.onOptionsItemSelected(item);
}
```

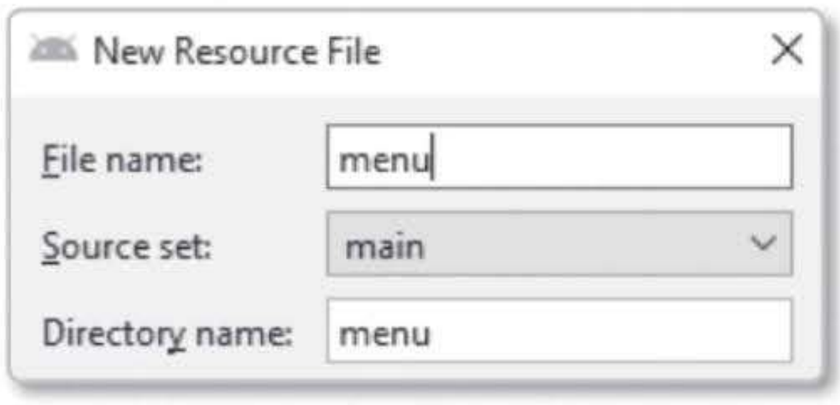
Por ejemplo, para desarrollar un menú pop up, podemos seguir los siguiente pasos:

1. Creamos un contenedor de menú presionando clic derecho en el proyecto **> new > Android Resource Directory >.**

2. Asignamos las siguientes opciones:



3. Notaremos que se ha creado una carpeta **Menú** dentro del paquete **res** de la aplicación. Aquí presionamos clic derecho **> new > Menú resource file** y asignamos un nombre.



4. Colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<menu xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'>
    <item
        android:icon='@drawable/ic_launcher_background'
        android:title='Archivo'
        android:id='@+id/archivo'
        app:showAsAction='always'
        />
```

```
        <item
            android:icon="@drawable/ic_launcher_background"
            android:title="Edicion"
            android:id="@+id/edicion"
            app:showAsAction="always"
            />
        <item
            android:icon="@drawable/ic_launcher_background"
            android:title="Formato"
            android:id="@+id/formato"
            app:showAsAction="always"
            />
    </menu>
```

5. Agregamos un botón en el archivo **activity_main.xml**, de tal forma que el código se muestre de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btMostrar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mostrar Popup" />
</android.support.constraint.ConstraintLayout>
```

6. Invocamos el menú en el activity, que lo mostrará usando el siguiente código (Henao, 2017):

```
package com.torres.manuel.myapplication;

import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.PopupMenu;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
                          implements PopupMenu.OnMenuItemClickListener {

    Button btMostrar;

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

        btMostrar=(Button)findViewById(R.id.btMostrar);
        btMostrar.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(MainActivity.this,v);
                popup.setOnMenuItemClickListener(MainActivity.this);
                popup.inflate(R.menu.menu);
                popup.show();
            }
        });

    }

    @Override
    public boolean onMenuItemClick(MenuItem item){
        switch (item.getItemId()){
            case R.id.archivo:
                Toast.makeText(this,'Ha seleccionado Archivo',.
                                                Toast.LENGTH_SHORT).show();
                return true;
                case R.id.edicion:
                Toast.makeText(this,'Ha seleccionado Edicion',
                                                Toast.LENGTH_SHORT).show();
                    return true;
                case R.id.formato:
                Toast.makeText(this,'Ha seleccionado Formato',
                                                Toast.LENGTH_SHORT).show();
                    return true;
            }
            return false;
        }
    }
```

CASO DESARROLLADO 01: **manejo de LinearLayout horizontal**

Implementemos una aplicación móvil que permita registrar los datos de un determinado cliente. Estos deberán pasar a otra pantalla usando el LinearLayout con orientación horizontal. Así mismo, debemos pasar información de una pantalla a otra de la aplicación.

Inicialmente, la aplicación debe mostrarse de la siguiente manera:



Al ingresar los datos en los controles y seleccionar el botón Registrar, se deberá enviar dicha información a otra pantalla de la siguiente manera:



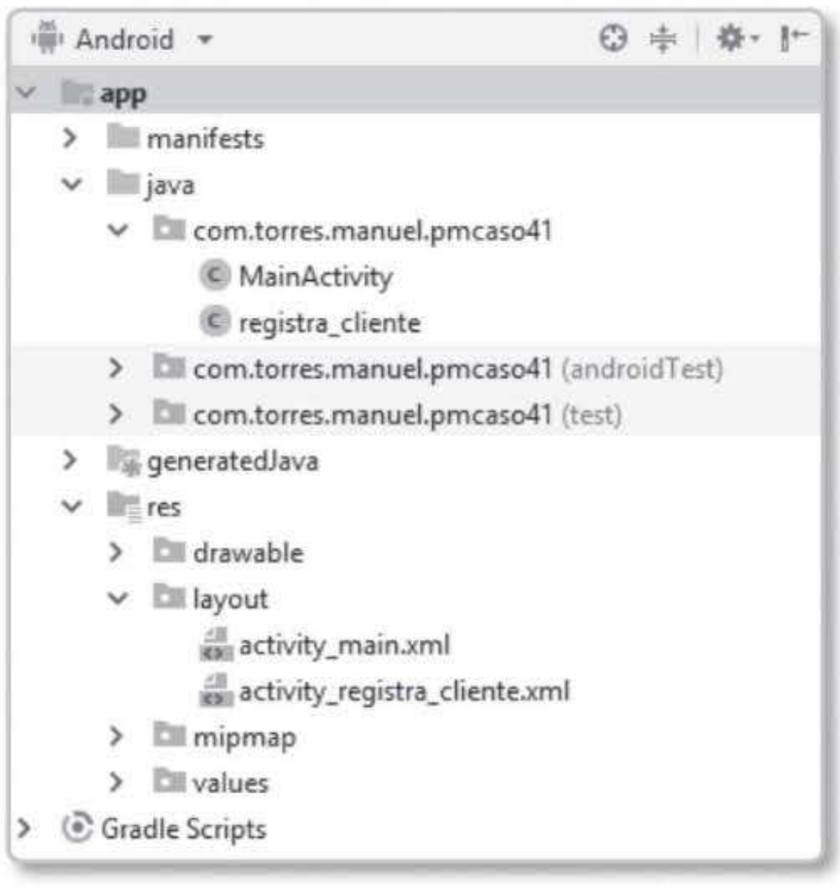
Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 38):

Tabla 38. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso41 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_04
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos un nuevo Layout al proyecto y presionamos clic derecho sobre el proyecto (app) desde la ventana **Exploradora de Proyectos > New > Activity > Empty Activity**. Luego, asignamos el nombre registra_cliente y colocamos finalizar.
4. El proyecto debe mostrarse de la siguiente manera:



5. Diseñamos el prototipo de la aplicación mediante XML en el archivo **activity_main.xml** como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='horizontal'
    tools:context='.MainActivity'>
    <TextView
        android:id='@+id/textView'
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:layout_weight='1'
        android:textColor='#000'
        android:text='REGISTRO DE CLIENTES' />
    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:layout_weight='2'
        android:textColor='#000'
        android:orientation='vertical'>
        <EditText
            android:id='@+id/etNombres'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:ems='10'
            android:hint='Nombres'
            android:inputType='textPersonName' />

        <EditText
            android:id='@+id/etApellidos'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:ems='10'
            android:hint='Apellidos'
            android:inputType='textPersonName' />
        <EditText
            android:id='@+id/etEmail'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:ems='10'
            android:hint='Email'
            android:inputType='textEmailAddress' />
        <EditText
            android:id='@+id/etTelefono'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:ems='10'
            android:hint='Telefono'
            android:inputType='phone' />
        <EditText
            android:id='@+id/etDireccion'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:ems='10'
            android:hint='Dirección'
            android:inputType='textPersonName' />
```

```
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="bottom"
            android:orientation="vertical">
            <Button
                android:id="@+id/btRegistrar"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="REGISTRAR" />
            <Button
                android:id="@+id/btCancelar"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="CANCELAR" />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

6. Agregamos la lógica de la aplicación en el archivo **Main_Activity.java**.

```
package com.torres.manuel.pmcaso41;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener {

    //Declaración de objetos
    EditText etNombres,etApellidos,etDireccion,etTelefono, etEmail;
    Button btRegistrar,btCancelar;

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

        //Asociar los botones a los objetos
        etNombres=(EditText)findViewById(R.id.etNombres);
        etApellidos=(EditText)findViewById(R.id.etApellidos);
        etEmail=(EditText)findViewById(R.id.etEmail);
        etTelefono=(EditText)findViewById(R.id.etTelefono);
        etDireccion=(EditText)findViewById(R.id.etDireccion);

        btRegistrar=(Button) findViewById(R.id.btRegistrar);
        btRegistrar.setOnClickListener(this);
        btCancelar=(Button) findViewById(R.id.btCancelar);
        btCancelar.setOnClickListener(this);
    }

    @Override
    public vo id onClick(View v){
        if(v.getId()==R.id.btRegistrar){
```

```
            Intent intent=new Intent(this,registra_cliente.class);
            intent.putExtra('nombres',etNombres.getText().toString());
            intent.putExtra('apellidos',etApellidos.getText().toString());
            intent.putExtra('email',etEmail.getText().toString());
            intent.putExtra('telefono',etTelefono.getText().toString());
            intent.putExtra('direccion',etDireccion.getText().toString());
            startActivity(intent);
        }
        else if(v.getId()==R.id.btCancelar){
            etNombres.setText('');
            etApellidos.setText('');
            etEmail.setText('');
            etTelefono.setText('');
            etDireccion.setText('');
            etNombres.requestFocus();
        }
    }
}
```

7. Diseñamos el prototipo de la pantalla que recibe la información del registro mediante XML en el archivo **activity_registra_cliente.xml**, tal como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout xmlns:android='http://
schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.registra_cliente'>
    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:padding='10dp'
        android:orientation='horizontal'>
        <TextView
            android:id='@+id/tvDatos'
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:textColor='#000'
            android:layout_weight='1' />
        <TextView
            android:id='@+id/textView5'
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:textColor='#000'
            android:layout_weight='1'
            android:text='RESUMEN DE CLIENTE' />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

8. Agregamos la lógica de la actividad que recibe la información en el archivo **registra_cliente.java**.

```
package com.torres.manuel.pmcaso41;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.EditText;
import android.widget.TextView;
public class registra_cliente extends AppCompatActivity {
```

```
    TextView tvDatos;

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

        tvDatos = (TextView) findViewById(R.id.tvDatos);

        Bundle datos=getIntent().getExtras();
        String nombres=datos.getString('nombres');
        String apellidos=datos.getString('apellidos');
        String email=datos.getString('email');
        String telefono=datos.getString('telefono');
        String direccion=datos.getString('direccion');

        String info = 'NOMBRES: '+nombres+'\n'
                +'APELLIDOS: '+apellidos+'\n'
                +'EMAIL: '+email+'\n'
                +'TELEFONO: '+telefono+'\n'
                +'DIRECCION: '+direccion;
        tvDatos.setText(info);
    }
}
```

CASO DESARROLLADO 02: **manejo de LinearLayout vertical**

Implementemos una aplicación móvil que permita crear el diseño de inicio de sesión usando LinearLayout con orientación vertical. Inicialmente, la aplicación debe mostrarse de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 39):

Tabla 39. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso42 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_04
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**.

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:orientation='vertical'
        android:padding='40dp'
        tools:ignore='MissingConstraints'>

        <TextView
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:layout_gravity='center_horizontal'
            android:layout_weight='0'
            android:text='Iniciar Sesión'
            android:textAppearance='?android:attr/textAppearanceLarge' />

        <EditText
            android:id='@+id/txtUsuario'
            android:layout_width='match_parent'
            android:layout_height='0dp'
            android:layout_gravity='center_horizontal'
            android:layout_weight='1'
            android:hint='Usuario' />

        <EditText
            android:id='@+id/txtClave'
            android:layout_width='match_parent'
            android:layout_height='0dp'
            android:layout_gravity='center_horizontal'
            android:layout_weight='1'
            android:ems='10'
            android:hint='Clave'
            android:inputType='textPassword' />
```

```
        <Button
            android:id="@+id/btnAcceder"
            style="@style/TextAppearance.AppCompat.Button"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_gravity="center_horizontal"
            android:layout_weight="1"
            android:text="Acceder" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="0dp"
            android:layout_gravity="center_horizontal"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Si tiene algún problema llame al #103"
            android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
            android:textColor="#0E8AEE" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

4. Como esta aplicación solo es para diseñar un prototipo, no realizamos "modificar" en el archivo **Main_Activity.java**.

CASO DESARROLLADO 03: manejo de TableLayout

Implemente una aplicación móvil que permita crear el diseño que permita mostrar información de las evaluaciones de unos estudiantes. La aplicación tiene por objetivo concentrarnos en las posiciones de los controles. Inicialmente, esta debe mostrarse de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 40):

Tabla 40. Especificación de valores para el proyecto

Create Android Project	Application Name	: pmCaso43
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_04

Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:stretchColumns="1">

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="20dp"
            android:gravity="center"
            android:text="CONTROL DE EVALUACIONES"
            android:textColor="#000000" />

        <TableRow android:background="#000000">
            <TextView
                android:layout_column="0"
                android:padding="3dp"
                android:text="ESTUDIANTE"
                android:textColor="@android:color/white" />
            <TextView
                android:layout_column="3"
                android:padding="1dp"
                android:text="PROMEDIO"
                android:textColor="@android:color/white" />
        </TableRow>

        <TableRow>

            <TextView
                android:layout_column="0"
                android:padding="3dp"
                android:text="Carlos Hurtado Rojas"
                android:textColor="#000000" />
            <TextView
                android:layout_column="3"
                android:gravity="left"
                android:textColor="#000000"
                android:layout_gravity="right"
                android:text="17.50" />
        </TableRow>
```

```
        <TableRow>
            <TextView
                android:layout_column="0"
                android:padding="3dp"
                android:textColor="#000000"
                android:text="Alba Oviedo Cardenas" />
            <TextView
                android:layout_column="3"
                android:gravity="left"
                android:textColor="#000000"
                android:layout_gravity="right"
                android:text="15.60" />
        </TableRow>

        <TableRow>
            <TextView
                android:layout_column="0"
                android:padding="3dp"
                android:textColor="#000000"
                android:text="Guadalupe Acosta Ferrer" />
            <TextView
                android:layout_column="3"
                android:gravity="left"
                android:textColor="#000000"
                android:layout_gravity="right"
                android:text="20.00" />
        </TableRow>
        <View
            android:layout_height="7dp"
            android:background="#000000" />

        <TableRow>
            <TextView
                android:layout_column="1"
                android:padding="3dp"
                android:textColor="#000000"
                android:text="Máximo Promedio" />
            <TextView
                android:layout_column="3"
                android:gravity="left"
                android:textColor="#000000"
                android:layout_gravity="right"
                android:text="20.00" />
        </TableRow>

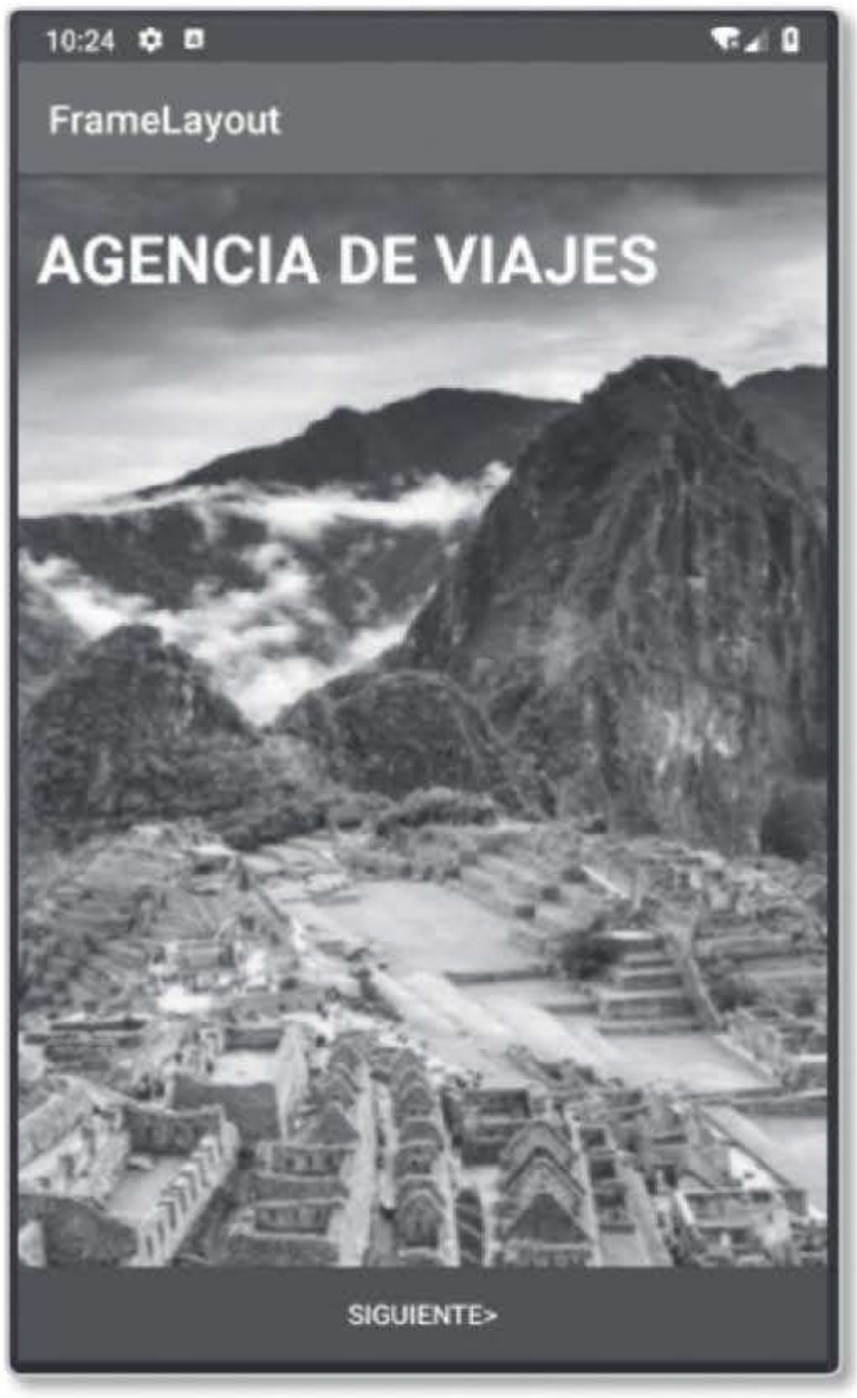
        <TableRow>
            <TextView
                android:layout_column="1"
                android:padding="3dp"
                android:textColor="#000000"
                android:text="Mínimo Promedio" />
            <TextView
                android:layout_column="3"
                android:gravity="left"
                android:layout_gravity="right"
                android:textColor="#000000"
                android:text="15.60" />
        </TableRow>
    </TableLayout>


</android.support.constraint.ConstraintLayout>
```

4. Como esta aplicación solo se emplea para diseñar un prototipo, no realizamos "modificar" en el archivo **Main_Activity.java**.

CASO DESARROLLADO 04: manejo de FrameLayout

Implementemos una aplicación móvil que permita crear el diseño que muestre información de una agencia de viajes. La aplicación tiene por objetivo concentrarnos en las posiciones de los controles dentro de una FrameLayout. Inicialmente, la aplicación debe mostrarse de la siguiente manera:



Solución:

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 41):

Tabla 41. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso44 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_04
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**:

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <FrameLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:orientation='vertical'>

        <ImageView
            android:id='@+id/imageView4'
            android:layout_width='match_parent'
            android:layout_height='match_parent'
            android:scaleType='fitXY'
            app:layout_constraintStart_toStartOf='parent'
            app:srcCompat='@drawable/fondo' />

        <TextView
            android:id='@+id/textView'
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:layout_marginLeft='10dp'
            android:layout_marginTop='20dp'
            android:fontFamily='Tahoma'
            android:text='AGENCIA DE VIAJES'
            android:textColor='#ffffff'
            android:textSize='34dp'
            android:textStyle='bold' />

        <Button
            android:id='@+id/button'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:layout_gravity='bottom'
            android:background='#0000ff'
            android:text='Siguiente>'
            android:textColor='#ffffff' />
    </FrameLayout>

</android.support.constraint.ConstraintLayout>
```

4. Como esta aplicación solo se emplea para diseñar un prototipo, no realizamos "modificar" en el archivo **Main_Activity.java**.

CASO DESARROLLADO 05: manejo de la clase CheckBox

Implementemos una aplicación móvil en Android que permita mostrar una serie de actividades que un usuario puede realizar en su tiempo libre. Este podrá seleccionar entre una y más opciones, tal como se muestra en la siguiente imagen:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 42):

Tabla 42. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso45 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_04
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos un nuevo Layout al proyecto y presionamos clic derecho sobre el proyecto (app) desde la ventana **Exploradora de Proyectos > New > Activity > Empty Activity**. Luego, asignamos el nombre registra_cliente, y colocamos finalizar.
4. Modificamos el código del archivo **activity_main.xml** de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <LinearLayout
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:orientation='vertical'
        android:id='@+id/contenedor'
        tools:layout_editor_absoluteX='8dp'
        tools:layout_editor_absoluteY='8dp'
        tools:ignore='MissingConstraints'>
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

5. Modificamos el código del **archivo main_activity.java** de la siguiente manera:

```
package com.torres.manuel.pmcaso41;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.LinearLayout;
import android.widget.Toast;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    //Declaracion GLOBAL
    LinearLayout contenedor;

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

        //Asociar el objeto al control
        contenedor = (LinearLayout) findViewById(R.id.contenedor);

        //Definir un arreglo de Hobbies
        ArrayList<check> lista=new ArrayList<check>();
        lista.add(new check(1,'Internet'));
        lista.add(new check(2,'Leer'));
        lista.add(new check(3,'Futbol'));
        lista.add(new check(4,'Programar'));

        //Definir las propiedades del control Check
        for(check c : lista){
            CheckBox cb=new CheckBox(getApplicationContext());
            cb.setText(c.nombre);
            cb.setId(c.codigo);
            cb.setTextColor(Color.BLUE);
            cb.setTextSize(20);
            contenedor.addView(cb);

            cb.setOnCheckedChangeListener(new
                                CompoundButton.OnCheckedChangeListener()
            {
                @Override
                public void onCheckedChanged(CompoundButton buttonView,
                                             boolean isChecked) {
                    Toast.makeText(getApplicationContext(),'Opcion :'+
                            buttonView.getId(),Toast.LENGTH_SHORT).show();
                }
            });
        }
    }

    //Crear la clase de tipo Check
    class check{
        public int codigo;
        public String nombre;

        public check(int codigo, String nombre) {
            this.codigo = codigo;
            this.nombre = nombre;
        }
    }
}
```

CAPÍTULO

5

Controles avanzados para aplicaciones Android



5.1 Orientación del dispositivo

Dentro de las miles de aplicaciones móviles vamos a encontrar que muchos de estas rotan el entorno de acuerdo con el movimiento que se realiza en el dispositivo, pero hay algunas de ellas que por requerimiento de la aplicación no deberían rotar o deberían simplemente mantenerse en una determinada orientación (ver Figura 32).

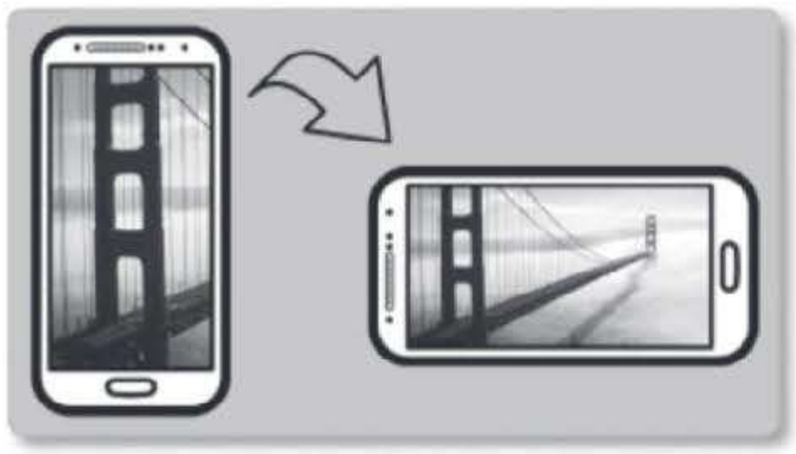


Figura 32. Orientación del dispositivo. Tomado de "¿Cómo controlar la orientación vertical en aplicaciones?," por Verlayne RGL, 2019. Recuperado de https://galaxyandroid.com/controlar-orientacion-vertical/.

La especificación de orientación del dispositivo se puede realizar en el archivo **AndroidManifest.xml** y por medio del archivo activity de Java, pero aquí debemos importar la librería import **android.content.pm.ActivityInfo**. Veamos las siguientes casos de orientación (ver Tabla 42).

Tabla 42. Código de orientación de dispositivo

Orientación vertical desde AndroidManifest	android:**screenOrientation**="landscape"
Orientación horizontal desde AndroidManifest	android:**screenOrientation**="portrait"
Orientación vertical desde Java	**this**.setRequestedOrientation(ActivityInfo .SCREEN_ORIENTATION_PORTRAIT);
Orientación horizontal desde Java	**this**.setRequestedOrientation(ActivityInfo .SCREEN_ORIENTATION_LANDSCAPE);

Para realizar el bloqueo de orientación en el dispositivo, podemos usar los siguientes códigos en XML (ver Tabla 43):

Tabla 43. Código de bloqueo de orientación del dispositivo

Bloqueo de orientación vertical	**this**.setRequestedOrientation(ActivityInfo .SCREEN_ORIENTATION_LANDSCAPE);
Bloqueo de orientación horizontal	**this**.setRequestedOrientation(ActivityInfo .SCREEN_ORIENTATION_PORTRAIT);

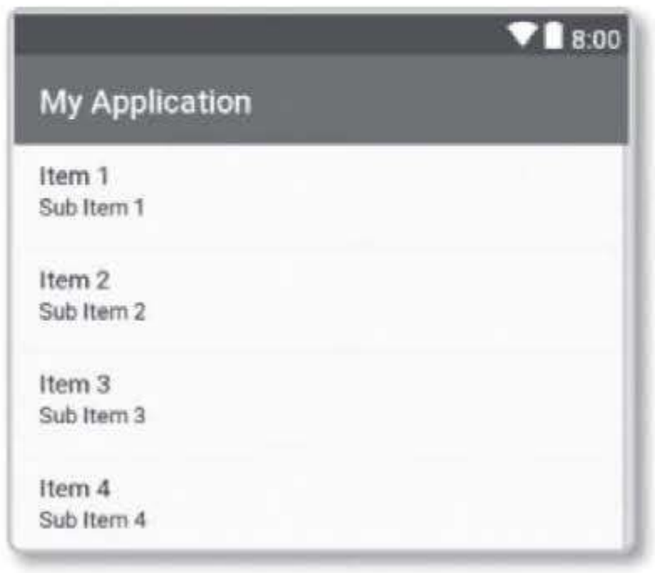
5.2 Control ListView

Permite visualizar una lista de elementos con deslizamiento vertical. Para poder implementar un ListView, primero, debemos definir el modelo de lista mediante un Layout. Este modelo será repetido en el contenedor final, y finalmente debemos asociarlo en un Layout principal. Se recomienda manejar las opciones que se muestran en el ListView mediante un arreglo de opciones o mediante una tabla de base de datos.

Inicialmente, el ListView en código XML se muestra de la siguiente manera:

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Y en el diseño se muestra del siguiente modo:



Entre sus principales propiedades se encuentran las señaladas en la Tabla 44:

Tabla 44. Propiedades XML para ListView.

PROPIEDAD	DESCRIPCIÓN
android:background	Especifica un color de fondo a la lista, por ejemplo, podemos asignar el color android:background="@color/colorPrimaryDark" o también podríamos especificarlo por medio del código hexadecimal android:background="#FF0"
android:clickable	Activa si los elementos accionan al presionar un clic. Para esto, la propiedad cuenta con false para desactivar y true para habilitar el clic. Por ejemplo, android:clickable="true".
android:divider	Define un color de la línea divisoria entre los elementos de la lista. Por ejemplo, android:divider="@color/colorPrimary".
android:dividerHeight	Define el ancho de la línea horizontal que divide los elementos de la lista. Por ejemplo, android:dividerHeight="5dp" define 5dp de grosor de la línea.
android:id	Permite asignar un nombre al control ListView de tal forma que pueda ser asociado dentro del código Java. Por ejemplo, android:id="@+id/lvArticulos", el cual define un ListView para productos.

Por ejemplo, si queremos mostrar una lista de productos de la siguiente forma...



Empezamos por definir el diseño en XML del siguiente modo:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:padding='20dp'
        android:textColor='#000'
        android:textAlignment='center'
        android:textSize='20dp'
        android:text='LISTADO DE ARTICULOS' />

    <ListView
        android:id='@+id/lvArticulos'
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:dividerHeight='10dp'
        />
</LinearLayout >
```

Luego, en Java, definimos un arreglo de artículos y lo asignamos en el control ListView:

```
package com.torres.manuel.myapplication;

import android.content.pm.ActivityInfo;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    //Declaracion de objeto de tipo ListView
    ListView lvArticulos;

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

        //Definir los valores del ListView
        String[] articulos=new String[]{'Lavadora 15p','Televisor 42 pulg.',
                                        'Radiograbadora','DVD'};

        //Asociar el objeto al control
        lvArticulos = (ListView)findViewById(R.id.lvArticulos);

        //Creamos un adaptador para enviar los valores a mostrar
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                            android.R.layout.simple_list_item_1, articulos);
```

```
        lvArticulos.setAdapter(adapter);

        //Mostrar un mensaje al seleccionar un elemento de la lista
    lvArticulos.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                                                    int position, long id) {
                int pos = position;
               String elemento=(String)lvArticulos.getItemAtPosition(position);
                Toast.makeText(getApplicationContext(), 'Position: '
                        + pos+' - Valor: '+elemento, Toast.LENGTH_LONG).
show();
            }
        });
    }
}
```

5.3 Control CardView

Es un tipo especial de Layuout que permite mostrar información tipo tarjetas, la cual puede incluir imágenes y textos con un aspecto de portada como un afiche. Los controles que incluye un CardView se superponen uno sobre otro, pero se pueden administrar mediante propiedades.

Inicialmente, debemos tener en cuenta que el uso de los CardView necesita un soporte, el cual se puede aplicar en el archivo **build.gradle(module_app)** colocando el código **> compile 'com.android.support:cardview-v7:28.0.0'** dentro de **dependencies**.

En la Tabla 45, se aprecian sus principales propiedades.

Tabla 45. Propiedades XML para CardView

PROPIEDAD	DESCRIPCIÓN
android:**layout_width**	Definición de ancho del CardView.
android:**layout_height**	Definición del alto del CardView.
app:**cardCornerRadius**	Define un valor de redondeo de las esquinas del CardView.
android:**layout_margin**	Define los márgenes del CardView sobre el contenedor actual.

Por ejemplo, si queremos implementar el siguiente modelo de CardView:



Agregamos la dependencia de tal forma que el archivo **build.gradle** se muestra de la siguiente forma:

```
    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        implementation 'com.android.support:appcompat-v7:28.0.0'
        implementation 'com.android.support.constraint:constraint-layout:1.1.3'
        implementation 'com.android.support:cardview-v7:28.0.0'
        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'
    }
```

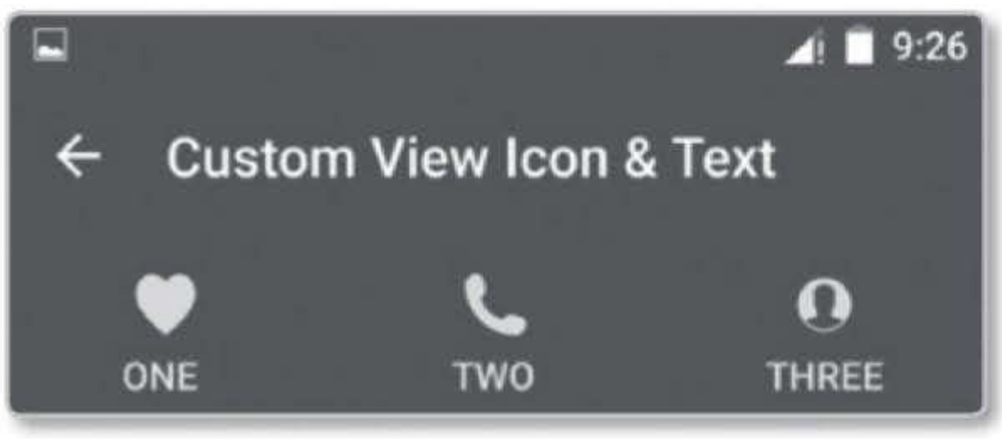
El diseño se muestra en el siguiente código XML:

```
    <?xml version='1.0' encoding='utf-8'?>
    <LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
        xmlns:app='http://schemas.android.com/apk/res-auto'
        xmlns:tools='http://schemas.android.com/tools'
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:orientation='vertical'
        tools:context='.MainActivity'>
        <android.support.v7.widget.CardView
            android:layout_width='match_parent'
            android:layout_height='190dp'
            app:cardCornerRadius='10dp'
            android:layout_margin='20dp'>
            <LinearLayout
                android:layout_width='match_parent'
                android:layout_height='match_parent'
                android:background='#0000FF'
                android:orientation='horizontal'>
                <LinearLayout
                    android:layout_width='match_parent'
                    android:layout_height='match_parent'
                    android:layout_margin='30dp'
                    android:orientation='horizontal'>
                    <LinearLayout
                        android:layout_width='wrap_content'
                        android:layout_height='match_parent'
                        android:padding='10dp'
                        android:orientation='vertical'>
                        <TextView
                            android:id='@+id/textView'
                            android:layout_width='match_parent'
                            android:layout_height='wrap_content'
                            android:textSize='30dp'
                            android:textColor='#fff'
                            android:text='Lima-Perú' />
                        <TextView
                            android:id='@+id/textView2'
                            android:layout_width='match_parent'
                            android:layout_height='wrap_content'
                            android:textColor='#fff'
                            android:text='Sábado, 12:00-Nublado' />
                        <TextView
                            android:id='@+id/textView3'
                            android:layout_width='match_parent'
                            android:layout_height='wrap_content'
                            android:textColor='#fff'
                            android:textSize='40dp'
                            android:text='22°C' />
```

```
            </LinearLayout>
            <LinearLayout
                android:layout_width='match_parent'
                android:layout_height='match_parent'
                android:orientation='vertical'>
                <ImageView
                    android:id='@+id/imageView2'
                    android:layout_width='match_parent'
                    android:layout_height='match_parent'
                    app:srcCompat='@drawable/nube' />
            </LinearLayout>
        </LinearLayout>
    </android.support.v7.widget.CardView>
</LinearLayout>
```

5.4 Control TabLayout

Permite diseñar una aplicación basada en pestañas, de tal forma que podemos organizar de la mejor manera posible los componentes móvil. Su implementación se basa en vistas que son controladas por las pestañas del TabLayout.



En la Tabla 46, se muestran sus principales propiedades.

Tabla 46. Propiedades XML para TabLayout

PROPIEDAD	DESCRIPCIÓN
android:**layout_width**	Define el ancho del TabLayout sobre su contenedor; normalmente, se le asigna match_parent.
android:**layout_height**	Define el alto del TabLayout sobre su contenedor; normalmente, se le asigna wrap_content.
app:**tabGravity**	Define la posición de las fichas del TabLayout; normalmente, se le asigna "fill" para rellenar la ficha sobre su contenedor.
app:**tabMode**	tabMode = "fixed": Las pestañas del TabLayout se muestran todas simultáneamente. tabMode = "scrollable": Normalmente es usado cuando cierta cantidad de pestañas no caben dentro del aplicativo. Se emplean para contextos de navegación en interfaces táctiles cuando los usuarios no necesitan comparar directamente las etiquetas de pestaña.
android:**id**	Asigna un nombre al control TabLayout.
android:**background**	Define un color de fondo al TabLayout.
app:**tabTextColor**	Define el color del texto mostrado en la pestaña.

Inicialmente, al incorporar un TabLayout, se muestra el siguiente código:

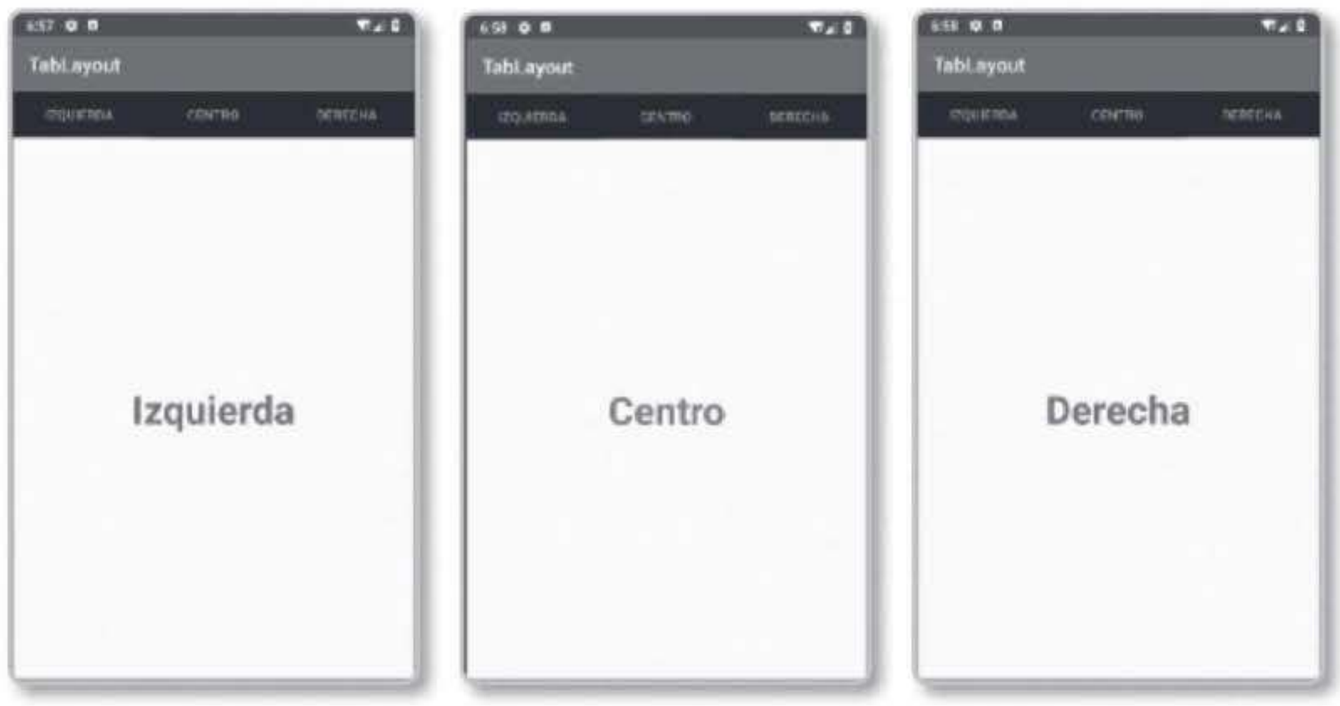
```
<android.support.design.widget.TabLayout
    android:layout_width='368dp'
    android:layout_height='21dp'>

    <android.support.design.widget.TabItem
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:text='Left' />

    <android.support.design.widget.TabItem
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:text='Center' />

    <android.support.design.widget.TabItem
        android:layout_width='wrap_content'
        android:layout_height='wrap_content'
        android:text='Right' />
</android.support.design.widget.TabLayout>
```

Vamos a implementar una aplicación que use el control TabLayout de la siguiente manera:



Primero, debemos agregar la sentencia **com.android.support:design:28.0.0** de la sección de dependencies del archivo **build.gradle (module:app)** de la aplicación:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    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'
    implementation 'com.android.support:design:28.0.0'
}
```

Segundo, diseñamos el prototipo en el archivo **activity_main.xml**:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <android.support.design.widget.TabLayout
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        app:tabGravity='fill'
        app:tabMode='fixed'
        android:id='@+id/tabLayout_id'
        android:background='@android:color/black'
        app:tabTextColor='@android:color/holo_blue_light' />

    <android.support.v4.view.ViewPager
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:id='@+id/viewpager_id'/>
</LinearLayout>
```

Tercero, agregamos un fragmento al proyecto (clic derecho sobre **app> new > Fragment > Fragment(Blank)**; asignamos el nombre izquierda_fragment.xml, y colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout
    xmlns:android='http://schemas.android.com/apk/res/android'
    android:layout_width='match_parent'
    android:layout_height='match_parent'>

    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:textSize='40dp'
        android:textStyle='bold'
        android:layout_gravity='center'
        android:gravity='center'
        android:text='Izquierda' />
</LinearLayout>
```

Cuarto, agregamos un fragmento al proyecto (clic derecho sobre **app> new > Fragment > Fragment(Blank)**; asignamos el nombre centro_fragment.xml y colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout
    xmlns:android='http://schemas.android.com/apk/res/android'
    android:layout_width='match_parent'
    android:layout_height='match_parent'>

    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:textSize='40dp'
        android:textStyle='bold'
        android:layout_gravity='center'
        android:textColor='@color/colorAccent'
        android:gravity='center'
        android:text='Centro' />
</LinearLayout>
```

Quinto, agregamos un fragmento al proyecto (clic derecho sobre **app> new > Fragment > Fragment(Blank)**; asignamos el nombre derecha_fragment.xml, y colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout
    xmlns:android='http://schemas.android.com/apk/res/android'
    android:layout_width='match_parent'
    android:layout_height='match_parent'>

    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:textSize='40dp'
        android:textStyle='bold'
        android:layout_gravity='center'
        android:gravity='center'
        android:text='Derecha' />
</LinearLayout>
```

Sexto, agregamos el siguiente código en el archivo **FragmentIzquierda.java**:

```
package com.torres.manuel.myapplication;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentIzquierda extends Fragment {
    View view;
    public FragmentIzquierda() {
    }

    @Nullable
    @Override
```

```
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable
ViewGroup container, @Nullable Bundle savedInstanceState) {
        view =inflater.inflate(R.layout.izquierda_fragment,container,false);
        return view;
    }
}
```

Séptimo, agregamos el siguiente código en el archivo **FragmentCentro.java**:

```
package com.torres.manuel.myapplication;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentCentro extends Fragment {

    View view;
    public FragmentCentro() {
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable
ViewGroup container, @Nullable Bundle savedInstanceState) {
        view =inflater.inflate(R.layout.centro_fragment,container,false);
        return view;
    }
}
```

Octavo, agregamos el siguiente código en el archivo **FragmentDerecha.java**:

```
package com.torres.manuel.myapplication;

import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentDerecha extends Fragment {
    View view;
    public FragmentDerecha() {
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable
ViewGroup container, @Nullable Bundle savedInstanceState) {
        view =inflater.inflate(R.layout.derecha_fragment,container,false);
        return view;
    }
}
```

Noveno, agregamos la clase **ViewPagerAdapter.java** en la carpeta Java del proyecto y colocamos el siguiente código:

```
package com.torres.manuel.myapplication;

import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import java.util.ArrayList;
import java.util.List;

public class ViewPagerAdapter extends FragmentPagerAdapter {

    private final List<Fragment> fragmentList = new ArrayList<>();
    private final List<String> fragmentListTitulos = new ArrayList<>();

    public ViewPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int i) {
        return fragmentList.get(i);
    }

    @Override
    public int getCount() {
        return fragmentListTitulos.size();
    }

    @Nullable
    @Override
    public CharSequence getPageTitle(int position) {
        return fragmentListTitulos.get(position);
    }

    public void addFragment(Fragment fragment, String titulo){
        fragmentList.add(fragment);
        fragmentListTitulos.add(titulo);
    }
}
```

Décimo, colocamos el siguiente código en el archivo **MainActivity.java**.

```
package com.torres.manuel.myapplication;

import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    TabLayout tabLayout;
    ViewPager viewPager;
```

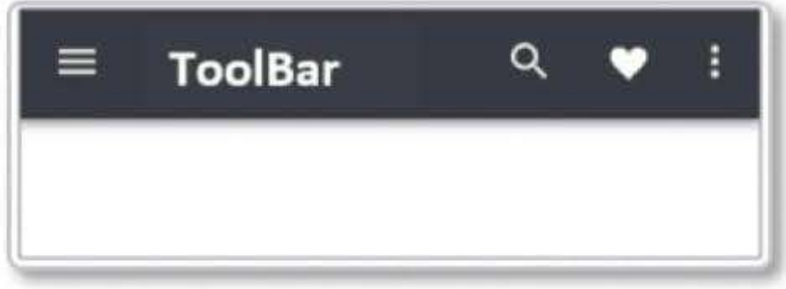
```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            tabLayout=(TabLayout)findViewById(R.id.tabLayout_id);
            viewPager=(ViewPager)findViewById(R.id.viewpager_id);
            ViewPagerAdapter adapter=new ViewPagerAdapter(getSupportFragmentManager());
            adapter.addFragment(new FragmentIzquierda(),'Izquierda');
            adapter.addFragment(new FragmentCentro(),'Centro');
            adapter.addFragment(new FragmentDerecha(),'Derecha');
            viewPager.setAdapter(adapter);
            tabLayout.setupWithViewPager(viewPager);

        }
    }
```

5.5 Control ToolBar

A partir de la versión Android 5.0, recibe el nombre de ToolBar; antes se llamaba ActionBar o AppBar. Es considerado como la barra de título o de herramientas. Comúnmente se muestra en la parte superior de la aplicación y puede presentar botones o imágenes de acción.



Un Toolbar se compone de un texto descriptivo, botones de acción y un menú overflow, a partir de los cuales podemos tomar el control de la aplicación. Veamos algunas propiedades del ToolBar en la Tabla 47.

Tabla 47. Propiedades XML para ToolBar

PROPIEDAD	DESCRIPCIÓN
android:**layout_width**	Define el ancho del ToolBar sobre su contenedor; comúnmente se le asigna match_parent.
android:**layout_height**	Define el alto del ToolBar sobre su contenedor; comúnmente se le asigna wrap_content.
app:**title**	Define el título que se mostrará en el ToolBar.
app:**titleTextColor**	Define el color del texto mostrado como título en el ToolBar.
android:**id**	Asigna un nombre al control ToolBar.
android:**background**	Define un color de fondo al ToolBar.
app:**logo**	Define un ícono que se mostrará al costado del título del ToolBar. Para poder agregar un ícono, se presiona clic derecho en el proyecto > Vector Asset y se selecciona un ícono desde la opción ClipArt; luego, se debe seleccionar un color para el ícono y, finalmente, asociarlo con la propiedad "app:logo".

Vamos a implementar una aplicación que use el control TabLayout de la siguiente manera:



Primero, debemos agregar al proyecto un archivo de tipo menú (clic derecho en **RES > New > Android Resource File > Seleccionar Resource Type=menú y asignar el nombre "menú"**); luego, colocaremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:title="Opción 1"
        android:id="@+id/opcion1"
        />
    <item
        android:title="Opción 2"
        android:id="@+id/opcion2"
        />
    <item
        android:title="Opción 3"
        android:id="@+id/opcion3"
        />
    <item
        android:title="Buscar"
        android:id="@+id/buscar"
        android:icon="@drawable/ic_search_amarillo"
        app:showAsAction="always"
        />
    <item
        android:title="Apagar"
        android:id="@+id/apagar"
        android:icon="@drawable/ic_settings_power_amarillo"
        app:showAsAction="always"
        />
</menu>
```

Segundo, diseñamos el prototipo de la aplicación en el archivo **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
```

```
            app:title="ToolBar"
            app:titleTextColor="#fff"
            app:logo="@drawable/ic_menu_amarillo" />
    </LinearLayout>
```

Tercero, colocamos el siguiente código java en el archivo **MainActivity.java**:

```
package com.torres.manuel.myapplication;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
import android.support.v7.widget.Toolbar;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

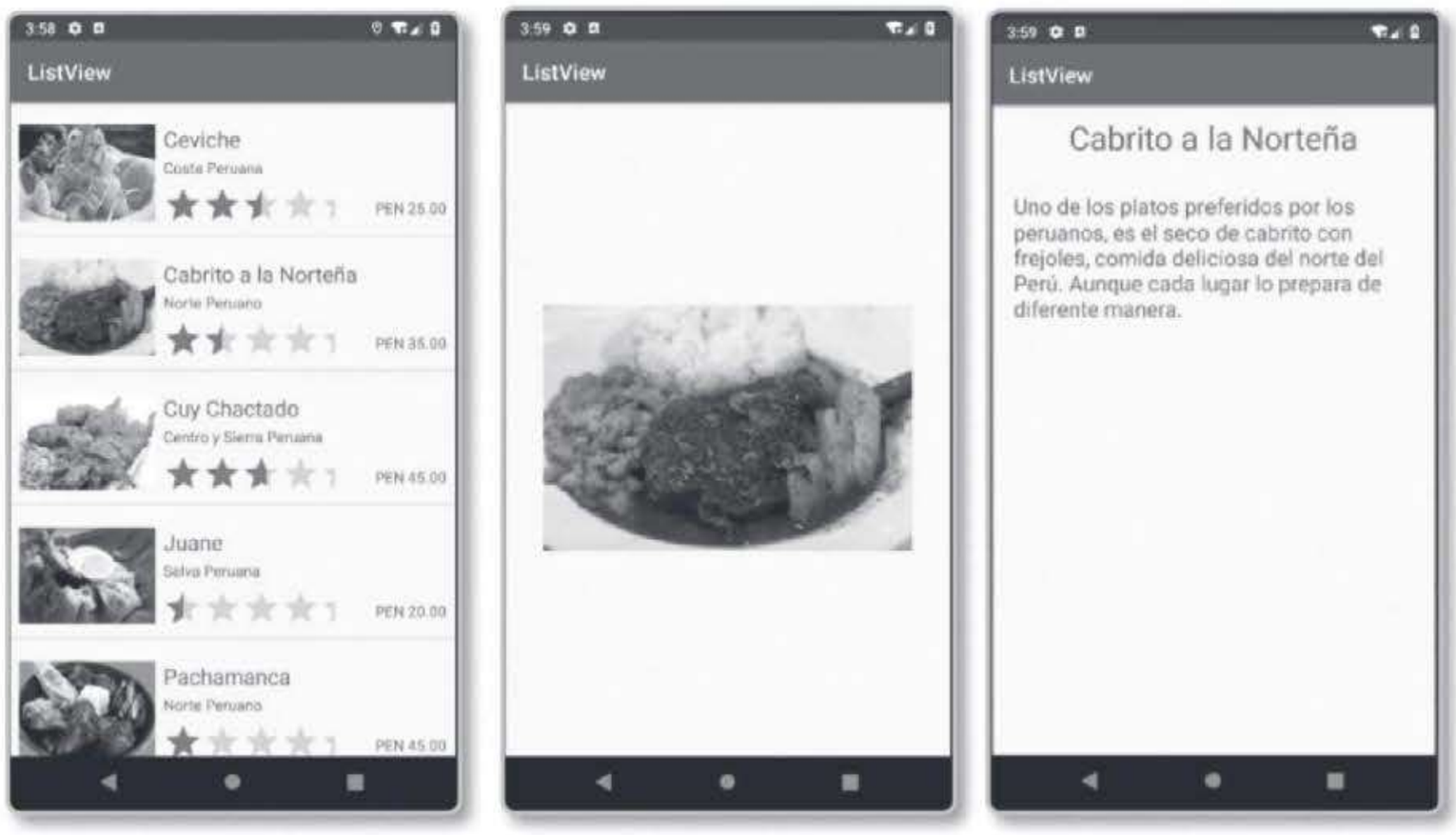
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id=item.getItemId();
        if(id==R.id.opcion1)
            Toast.makeText(this,"Opcion 1",Toast.LENGTH_SHORT).show();
        else if(id==R.id.opcion2)
            Toast.makeText(this,"Opcion 2",Toast.LENGTH_SHORT).show();
        else if(id==R.id.opcion3)
            Toast.makeText(this,"Opcion 3",Toast.LENGTH_SHORT).show();
        else if(id==R.id.buscar)
            Toast.makeText(this,"Buscar",Toast.LENGTH_SHORT).show();
        else if(id==R.id.apagar) {
            Toast.makeText(this, "Apagar", Toast.LENGTH_SHORT).show();
            Intent intent=new Intent(Intent.ACTION_MAIN);
            intent.addCategory(Intent.CATEGORY_HOME);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent);
        }
        return true;
    }
}
```

CASO DESARROLLADO 01:**manejo de ListView**

Implementemos una aplicación móvil que permita mostrar una lista de platos típicos peruanos y que, al seleccionar una imagen, se muestre en toda la pantalla, y si seleccionamos el título del plato, se muestre información del plato seleccionado. El manejo de la aplicación se muestra en las siguientes imágenes:



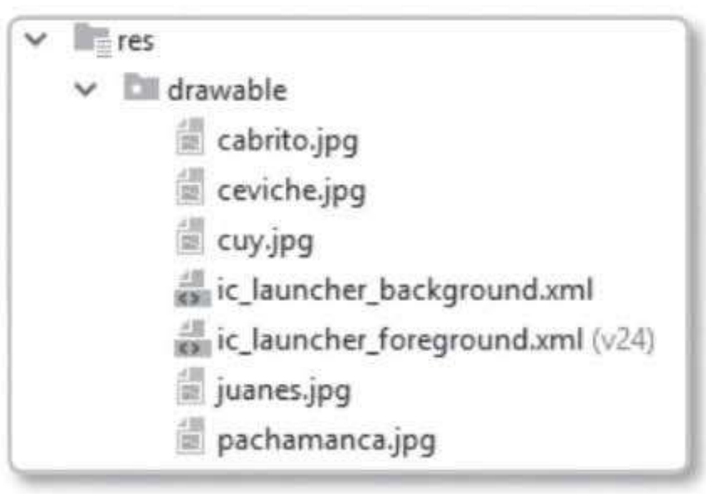
Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 48):

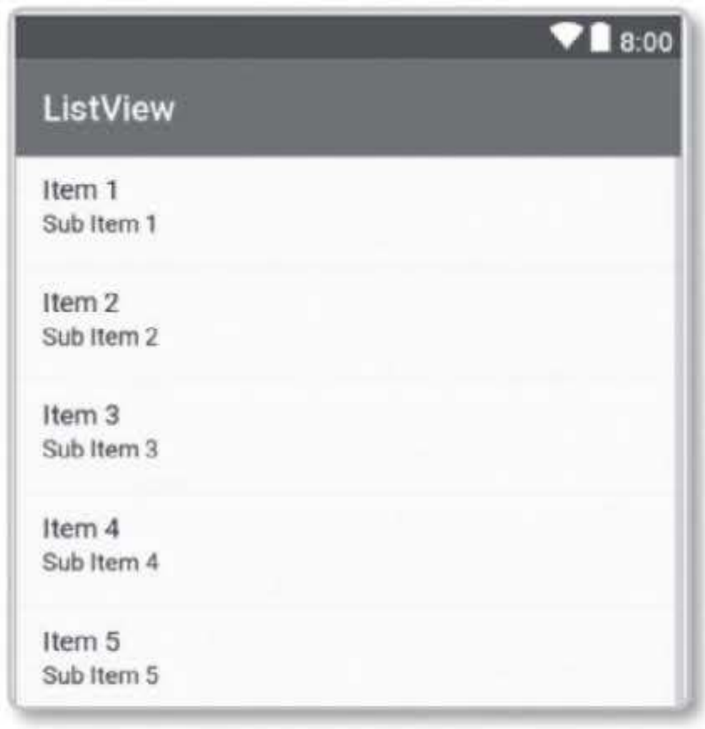
Tabla 48. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso51 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_05
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Ahora debemos copiar las imágenes que usaremos en el proyecto en la carpeta **drawable**. Esto se realizará pegando directamente en dicha carpeta, de tal forma que se muestre de la siguiente manera:



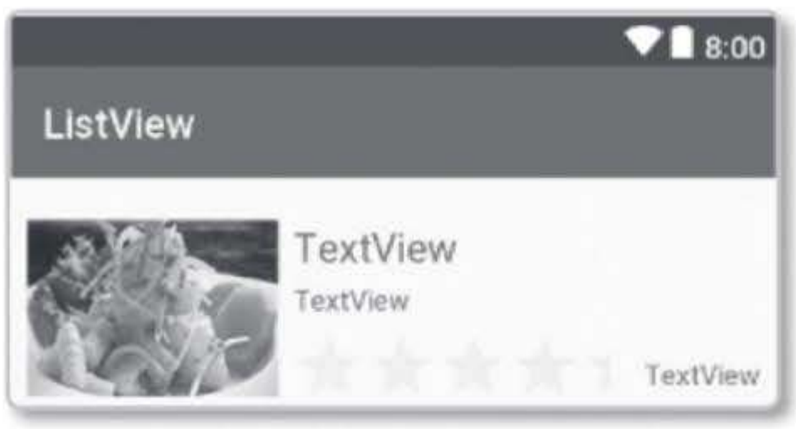
4. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**:




```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.MainActivity'>

    <ListView
        android:id='@+id/lvPlatos'
        android:layout_width='match_parent'
        android:layout_height='wrap_content' />
</android.support.constraint.ConstraintLayout>
```

5. Agregamos al proyecto un activity de tipo Empty Activity y le asignamos el nombre activity_elemento_lista.xml. Esto nos permitirá definir el aspecto que tendrán los platos en el ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='wrap_content'
    android:orientation='horizontal'
    tools:context='.elemento_lista'>
    <ImageView
            android:id='@+id/ivProducto'
            android:layout_width='126dp'
            android:layout_height='89dp'
```

```
            android:scaleType="fitXY"
            app:srcCompat="@drawable/lap01" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tvTitulo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView"
            android:textSize="20dp" />
        <TextView
            android:id="@+id/tvDescripcion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView" />
        <RatingBar
            android:id="@+id/rbPreferencia"
            style="@style/Widget.AppCompat.RatingBar.Indicator"
            android:layout_width="match_parent"
            android:layout_height="20dp" />
        <TextView
            android:id="@+id/tvCosto"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:text="TextView" />

    </LinearLayout>
</LinearLayout>
```

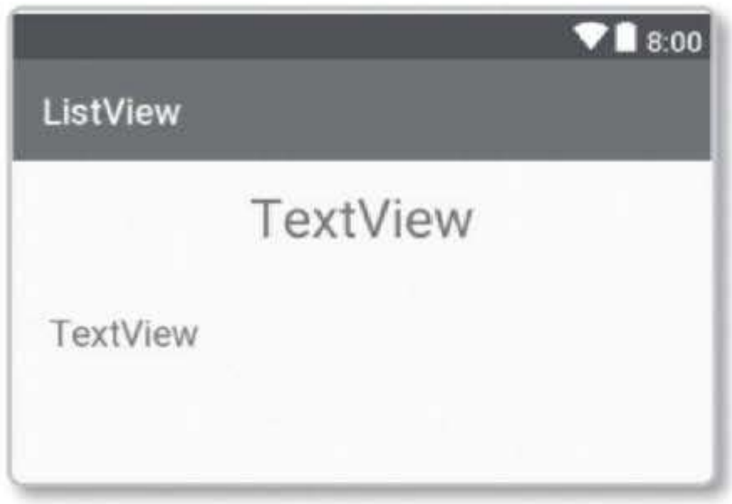
6. Agregamos al proyecto un activity de tipo Empty Activity, y le asignamos el nombre **activity_detalle_plato.xml**. Esto nos permitirá mostrar un texto descriptivo del plato seleccionado desde el control ListView.



```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.VisorImagenes'>

    <ImageView
        android:id='@+id/ivFotoCompleta'
        android:layout_width='341dp'
        android:layout_height='256dp'
        android:layout_marginStart='8dp'
        android:layout_marginTop='8dp'
        android:layout_marginEnd='8dp'
        android:layout_marginBottom='8dp'
        app:layout_constraintBottom_toBottomOf='parent'
        app:layout_constraintEnd_toEndOf='parent'
        app:layout_constraintStart_toStartOf='parent'
        app:layout_constraintTop_toTopOf='parent'
        app:layout_constraintVertical_bias='0.497'
        app:srcCompat='@drawable/cabrito' />
</android.support.constraint.ConstraintLayout>
```

7. Agregamos al proyecto un activity de tipo Empty Activity, y le asignamos el nombre activity_detalle_plato.xml. Esto nos permitirá mostrar un texto descriptivo del plato seleccionado desde el control ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.detalle_plato'>

    <TextView
        android:id='@+id/tvTituloDetalle'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='TextView'
        android:padding='10dp'
        android:gravity='center'
        android:textSize='30dp'
        />

    <TextView
```

```
        android:id="@+id/tvDetalle"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="TextView"
        android:layout_gravity="top"
        android:padding="20dp"
        android:textSize="20dp"
        />
</LinearLayout>
```

8. Agregamos la clase **Adaptador.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso51;

import android.content.Context;
import android.content.Intent;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.TextView;

public class Adaptador extends BaseAdapter {

    //Definir los objetos
    private static LayoutInflater inflater=null;

    Context context;
    String[][] informacion;
    int[] imagenes;

    //Metodo Constructor
public Adaptador(Context context, String[][] informacion, int[] imagenes) {
    this.context = context;
    this.informacion = informacion;
    this.imagenes = imagenes;
    inflater = (LayoutInflater)
            context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
    //Asociar la vista al activity_elemento_lista.xml
final View vista= inflater.inflate(R.layout.activity_elemento_lista,null);

    //Obtener todos los controles
    ImageView ivPlato= (ImageView)vista.findViewById(R.id.ivPlato);
    TextView tvTitulo= (TextView) vista.findViewById(R.id.tvTitulo);
    TextView tvOrigen= (TextView) vista.findViewById(R.id.tvOrigen);
    TextView tvPrecio= (TextView) vista.findViewById(R.id.tvPrecio);
RatingBar rbPreferencia=(RatingBar)vista.findViewById(R.id.rbPreferencia);

    //Asignar informacion a los controles por medio del arreglo
    tvTitulo.setText(informacion[position][0]);
    tvOrigen.setText(informacion[position][1]);
   rbPreferencia.setProgress(Integer.valueOf(informacion[position][2]));
    tvPrecio.setText(informacion[position][3]);
    ivPlato.setImageResource(imagenes[position]);

    //Metodo que para que la imagen acceda con Clic
    ivPlato.setTag(position);
    ivPlato.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
            public void onClick(View v) {
                Intent verFoto = new Intent(context,VisorImagenes.class);
                verFoto.putExtra('IMG',imagenes[(Integer)v.getTag()]);
                context.startActivity(verFoto);
            }
        });
        return vista;
    }

    //Metodo que devuelve la cantidad de platos
    @Override
    public int getCount() {
        return imagenes.length;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }
}
```

9. Agregamos la clase **VisorImagenes.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso51;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ImageView;

public class VisorImagenes extends AppCompatActivity {

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

        //Asociar la imagen al control
        ImageView ivFoto=(ImageView)findViewById(R.id.ivFotoCompleta);

        //Mostrar la imagen segun el plato seleccionado
        Intent intent=getIntent();
        Bundle b=intent.getExtras();
        if (b!=null) ivFoto.setImageResource(b.getInt('IMG'));
    }
}
```

10. Agregamos la clase **Detalle_Plato.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso51;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
```

```
public class detalle_plato extends AppCompatActivity {

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

        //Asociar los objetos a los controles
        TextView tvTituloDetalle=(TextView)findViewById(R.id.tvTituloDetalle);
        TextView tvDetalle=(TextView)findViewById(R.id.tvDetalle);

        //Muestra la actividad con parametros obtenidos
        Intent intent=getIntent();
        Bundle b=intent.getExtras();
        if (b!=null) {
            tvTituloDetalle.setText(b.getString('TIT'));
            tvDetalle.setText(b.getString('DET'));
        }
    }
}
```

11. En la clase **MainActivity.java**, colocamos el siguiente código:

```
package com.torres.manuel.pmcaso51;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    //Declarar el objeto ListView
    ListView lvPlatos;

    //Definicion de un arreglo de platos tipicos
    String[][] datos={
            {'Ceviche','Costa Peruana','5','PEN 25.00','Ceviche, que a
menudo se escribe ceviche o cebiche, dependiendo de qué parte de América
del Sur viene, es básicamente pescado crudo y mariscos marinados en jugo
de cítricos, principalmente jugo de limón. El cítrico del jugo coagula las
proteníinas del pescado, efectivamente cocinándolo, así no calentándolo, y
se sirve frío o a temperatura ambiente.'},
            {'Cabrito a la Norteña','Norte Peruano','3','PEN 35.00','Uno
de los platos preferidos por los peruanos, es el seco de cabrito con
frejoles, comida deliciosa del norte del Perú. Aunque cada lugar lo prepara
de diferente manera.'},
            {'Cuy Chactado','Centro y Sierra Peruana','6','PEN 45.00','El
cuy chactado es uno de los platos más populares en el Perú y que más adoran
sus habitantes debido a su sabor y las proteínas que este animal posee.'},
            {'Juane','Selva Peruana','1','PEN 20.00','El juane es uno de los
principales platos típicos de la gastronomía de la selva peruana y es muy
consumido durante la fiesta de San Juan que se celebra el 24 de junio de cada
año en honor a Juan Bautista. Tuvo su origen en la ciudad de Moyobamba.'},
            {'Pachamanca','Norte Peruano','2','PEN 45.00','La pachamanca
es un plato típico de la gastronomía del Perú, elaborado por la cocción,
al calor de piedras precalentadas, de carnes de vacuno, cerdo, pollo y
cuy previamente aderezados con ingredientes como chincho, huacatay, ají,
comino, pimiento y otras especias, junto con productos originales andinos
adicionales, como papas, camote, choclo, haba en vainas y, eventualmente,
yuca. Más allá de una comida o de un ágape, es un rito comunitario en el
Ande y en las ciudades.'},
```

```
        };

        //Definicion del arreglo de imagenes
        int[] imagenes={R.drawable.ceviche,R.drawable.cabrito,R.drawable.cuy,
                        R.drawable.juanes,R.drawable.pachamanca};

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

            //Asociar el objeto al control ListView
            lvPlatos = (ListView) findViewById(R.id.lvPlatos);
            //Asociar el adaptador al control ListView
            lvPlatos.setAdapter(new Adaptador(this,datos,imagenes));

            //Implementar el clic al seleccionar un titulo de la lista
            lvPlatos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view,
                                        int position, long id) {
                Intent visorDetalle=new Intent(view.getContext(),detalle_plato.class);
                    visorDetalle.putExtra('TIT',datos[position][0]);
                    visorDetalle.putExtra('DET',datos[position][4]);
                    startActivity(visorDetalle);
                }
            });
        }
    }
```

12. En el archivo **Strings.xml**, colocamos el siguiente código:

```
<resources>
    <string name='app_name'>ListView</string>
    <string name='title_activity_visor_imagenes'>VisorImagenes</string>
</resources>
```

13. El explorador del proyecto debe mostrarse de la siguiente manera:

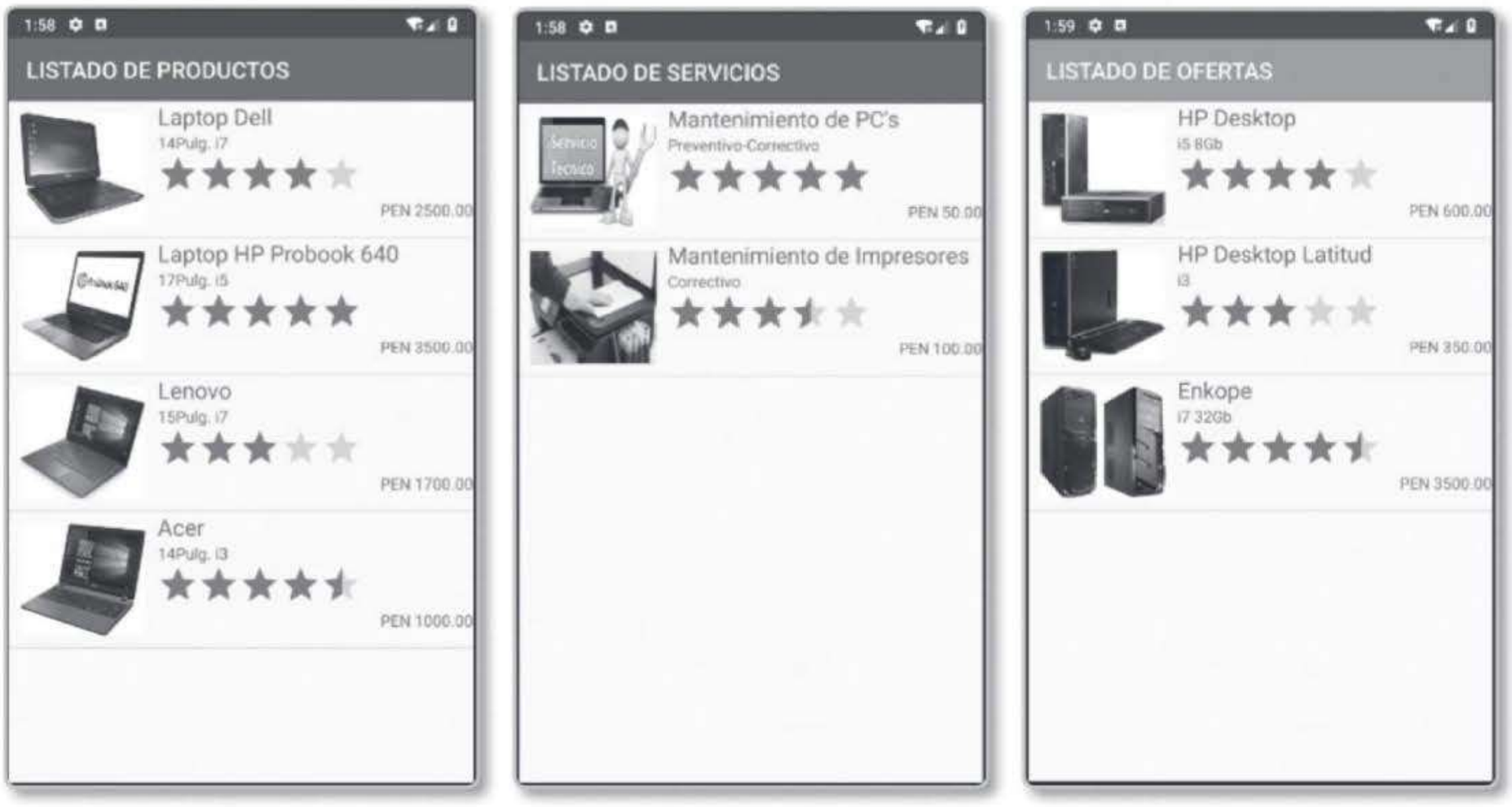


CASO DESARROLLADO 02: **manejo de CardView**

Implementemos una aplicación móvil que permita mostrar las tres líneas de negocio con que cuenta una empresa para que, al seleccionar una de estas, muestre, en una lista, los productos, los servicios y las ofertas de una empresa de computo. El manejo de la aplicación se muestra en las siguientes imágenes:



Hay que tener en cuenta que, al seleccionar un CardView, se debe mostrar la lista de productos, servicios u ofertas según la selección del usuario:



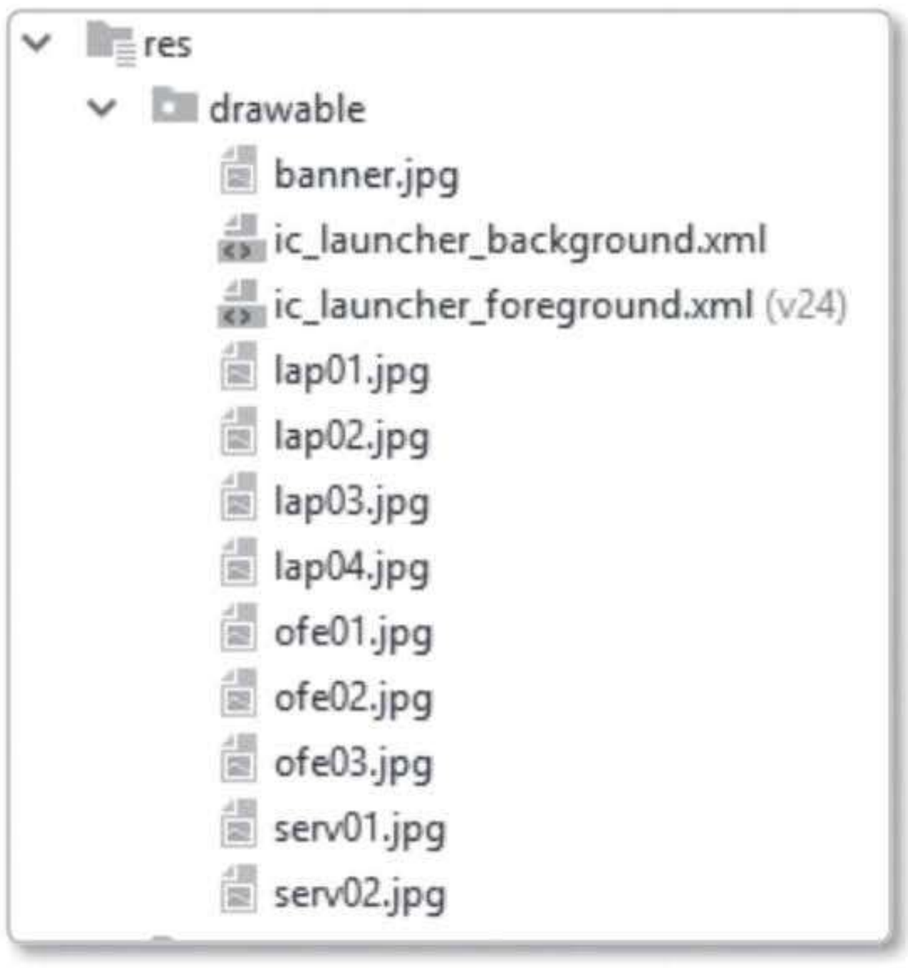
Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 49):

Tabla 49. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso52 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_05
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Ahora, debemos copiar las imágenes que usaremos en el proyecto en la carpeta **drawable**. Esto se realizará pegando directamente en dicha carpeta, de tal forma que se muestre de la siguiente manera:



4. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**.



```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    android:gravity='center'
    tools:context='.MainActivity'>

    <TextView
        android:id='@+id/textView'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='EMPRESA DE VENTAS Y SERVICIOS INFORMÁTICOS'
        android:textSize='20dp'
        android:gravity='center'
        android:textColor='#000'
        />

    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:gravity='center'
        android:orientation='horizontal'>

        <android.support.v7.widget.CardView
        android:layout_width='180dp'
        android:layout_height='220dp'
        android:layout_margin='10dp'
        android:id='@+id/cvProductos'>

        <LinearLayout
            android:layout_width='match_parent'
            android:layout_height='match_parent'
            android:gravity='top'
            android:orientation='vertical'>

            <ImageView
                android:id='@+id/ivBanner'
                android:layout_width='match_parent'
                android:layout_height='46dp'
                app:srcCompat='@drawable/banner' />

            <ImageView
                android:id='@+id/ivLap01'
                android:layout_width='50dp'
                android:layout_height='50dp'
                android:layout_marginTop='-20dp'
                android:layout_gravity='center'
                app:srcCompat='@drawable/lap01' />

            <TextView
                android:id='@+id/tvNuestrosProductos'
                android:layout_width='match_parent'
                android:layout_height='wrap_content'
                android:gravity='center'
                android:textColor='#000'
                android:textStyle='bold'
                android:text='Nuestros Productos' />
```

```
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <ImageView
                    android:id="@+id/ivLap01_1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/lap01" />

                <ImageView
                    android:id="@+id/ivLap03"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/lap03" />

                <ImageView
                    android:id="@+id/ivLap02"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/lap02" />
            </LinearLayout>
        </LinearLayout>
    </android.support.v7.widget.CardView>

    <android.support.v7.widget.CardView
        android:layout_width="180dp"
        android:layout_height="220dp"
        android:layout_margin="10dp"
        android:id="@+id/cvServicios">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="top"
            android:orientation="vertical">

            <ImageView
                android:id="@+id/ivBanner2"
                android:layout_width="match_parent"
                android:layout_height="46dp"
                app:srcCompat="@drawable/banner" />

            <ImageView
                android:id="@+id/ivServ01"
                android:layout_width="50dp"
                android:layout_height="50dp"
                android:layout_marginTop="-20dp"
                android:layout_gravity="center"
                app:srcCompat="@drawable/serv01" />

            <TextView
                android:id="@+id/tvNuestrosServicios"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
```

```
                android:textColor="#000"
                android:textStyle="bold"
                android:text="Nuestros Servicios" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <ImageView
                    android:id="@+id/ivServ01_1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/serv01" />

                <ImageView
                    android:id="@+id/ivServ02"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/serv02" />

            </LinearLayout>
        </LinearLayout>
    </android.support.v7.widget.CardView>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal">

    <android.support.v7.widget.CardView
        android:layout_width="180dp"
        android:layout_height="220dp"
        android:layout_margin="10dp"
        android:id="@+id/cvOfertas">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="top"
            android:orientation="vertical">

            <ImageView
                android:id="@+id/ivBanner3"
                android:layout_width="match_parent"
                android:layout_height="46dp"
                app:srcCompat="@drawable/banner" />

            <ImageView
                android:id="@+id/ivOfe01"
                android:layout_width="50dp"
                android:layout_height="50dp"
                android:layout_marginTop="-20dp"
                android:layout_gravity="center"
                app:srcCompat="@drawable/ofe01" />

            <TextView
```

```
                android:id="@+id/tvNuestrasOfertas"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:textColor="#000"
                android:textStyle="bold"
                android:text="Nuestras Ofertas" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">

                <ImageView
                    android:id="@+id/ivOfe01_1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/ofe01" />

                <ImageView
                    android:id="@+id/ivOfe02"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/ofe02" />

                <ImageView
                    android:id="@+id/ivOfe03"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    app:srcCompat="@drawable/ofe03" />
            </LinearLayout>
        </LinearLayout>
    </android.support.v7.widget.CardView>
</LinearLayout>
</LinearLayout>
```

5. Agregamos al proyecto un activity de tipo Empty Activity y le asignamos el nombre activity_nuestros_productos.xml. Esto nos permitirá definir el aspecto que tendrán los platos en el ListView.

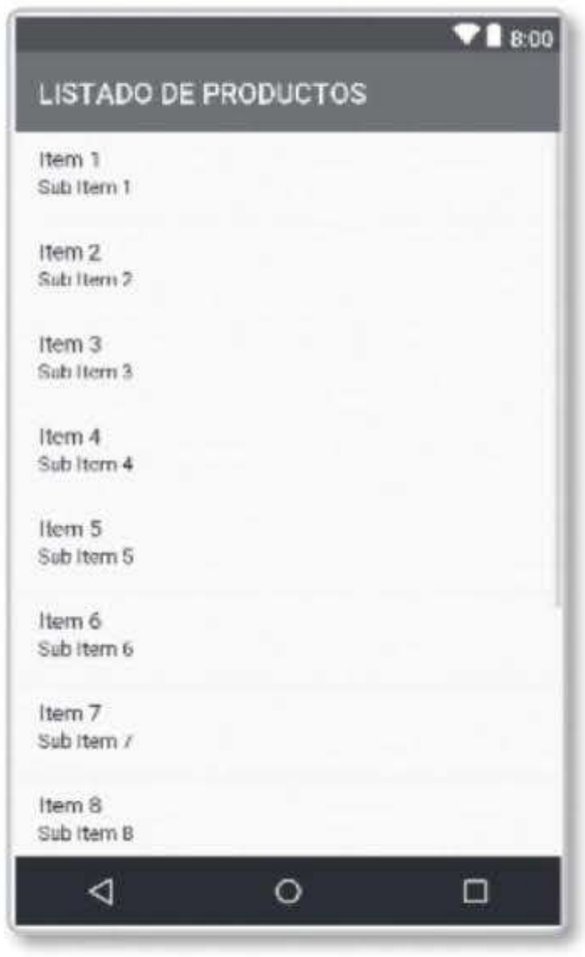


```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.nuestros_productos'>

    <android.support.v7.widget.Toolbar
        android:id='@+id/toolbar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:background='?attr/colorPrimary'
        android:minHeight='?attr/actionBarSize'
        app:title='LISTADO DE PRODUCTOS'
        app:titleTextColor='#FFF'
        android:theme='?attr/actionBarTheme' />

    <ListView
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:id='@+id/lvProductos'
        />
</LinearLayout>
```

6. Agregamos al proyecto un activity de tipo Empty Activity, y le asignamos el nombre activity_nuestros_servicios.xml. Esto nos permitirá definir el aspecto que tendrá la imagen al seleccionar una foto desde el control ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.nuestros_servicios'>
```

```
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#0066ff"
            android:minHeight="?attr/actionBarSize"
            app:title="LISTADO DE SERVICIOS"
            app:titleTextColor="#FFF"
            android:theme="?attr/actionBarTheme" />

        <ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/lvServicios"
            />
    </LinearLayout>
```

7. Agregamos al proyecto un activity de tipo Empty Activity y le asignamos el nombre activity_nuestras_ofertas.xml. Esto nos permitirá mostrar un texto descriptivo del plato seleccionado desde el control ListView.




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".nuestras_ofertas">
```

```
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#ff5500"
            android:minHeight="?attr/actionBarSize"
            app:title="LISTADO DE OFERTAS"
            app:titleTextColor="#FFF"
            android:theme="?attr/actionBarTheme" />

        <ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/lvOfertas"
            />
    </LinearLayout>
```

8. Agregamos al proyecto un activity de tipo Empty Activity y le asignamos el nombre activity_elemento_lista.xml. Esto nos permitirá mostrar un texto descriptivo del plato seleccionado desde el control ListView.




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    tools:context=".elemento_lista">
    <ImageView
        android:id="@+id/ivProducto"
        android:layout_width="132dp"
        android:layout_height="120dp"
        android:scaleType="fitXY"
        android:padding="10dp"
        app:srcCompat="@drawable/lap01" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tvTitulo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView"
            android:textSize="20dp" />
        <TextView
            android:id="@+id/tvDescripcion"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TextView" />
        <RatingBar
            android:id="@+id/rbPreferencia"
            style="@style/Widget.AppCompat.RatingBar.Indicator"
            android:layout_width="wrap_content"
            android:layout_height="40dp" />
        <TextView
            android:id="@+id/tvCosto"
            android:layout_width="wrap_content"
            android:layout_height="30dp"
            android:layout_gravity="right"
            android:text="TextView" />
    </LinearLayout>
</LinearLayout>
```

9. Agregamos la clase **Adaptador.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso52;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.TextView;

public class Adaptador extends BaseAdapter {

    //Definir los objetos
    private static LayoutInflater inflater=null;

    Context context;
    String[][] informacion;
    int[] imagenes;

    public Adaptador(Context context, String[][] informacion, int[] imagenes) {
        this.context = context;
        this.informacion = informacion;
        this.imagenes = imagenes;
        inflater = (LayoutInflater)
                context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public int getCount() {
        return imagenes.length;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
```

```
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            //Asociar la vista al activity_elemento_lista.xml
            final View vista= inflater.inflate(R.layout.activity_elemento_
    lista,null);

            //Obtener todos los controles
           ImageView ivProducto= (ImageView)vista.findViewById(R.id.ivProducto);
            TextView tvTitulo= (TextView) vista.findViewById(R.id.tvTitulo);
             TextView tvDescripcion= (TextView) vista.findViewById(R.
    id.tvDescripcion);
           TextView tvCosto= (TextView) vista.findViewById(R.id.tvCosto);
            RatingBar rbPreferencia=(RatingBar)vista.findViewById(R.
    id.rbPreferencia);

            //Asignar informacion a los controles por medio del arreglo
            tvTitulo.setText(informacion[position][0]);
            tvDescripcion.setText(informacion[position][1]);
             rbPreferencia.setProgress(Integer.valueOf(informacion[position]
    [2]));
            tvCosto.setText(informacion[position][3]);
            ivProducto.setImageResource(imagenes[position]);
            return vista;
        }
    }
```

10. Agregamos la clase **nuestros_productos.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso52;

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

public class nuestros_productos extends AppCompatActivity {

    ListView lvProductos;
    String[][] productos={{'Laptop Dell','14Pulg. i7','8','PEN 2500.00'},
                          {'Laptop HP Probook 640','17Pulg. i5','10','PEN
3500.00'},
                         {'Lenovo','15Pulg. i7','6','PEN 1700.00'},
                         {'Acer','14Pulg. i3','9','PEN 1000.00'}};

    int[]
imagenes={R.drawable.lap01,R.drawable.lap02,R.drawable.lap03,R.drawable.lap04};


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

        //Asociar el objeto al control ListView
        lvProductos = (ListView) findViewById(R.id.lvProductos);
        //Asociar el adaptador al control ListView
        lvProductos.setAdapter(new Adaptador(this,productos,imagenes));
    }
}
```

11. Agregamos la clase **nuestros_servicios.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso52;

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

public class nuestros_servicios extends AppCompatActivity {

    ListView lvServicios;
     String[][] servicios-{{'Mantenimiento de PC's','Preventivo-
Correctivo','10','PEN 50.00'},
          {'Mantenimiento de Impresores','Correctivo','7','PEN 100.00'}};

    int[] imagenes-{R.drawable.serv01,R.drawable.serv02};


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

        //Asociar el objeto al control ListView
        lvServicios - (ListView) findViewById(R.id.lvServicios);
        //Asociar el adaptador al control ListView
        lvServicios.setAdapter(new Adaptador(this,servicios,imagenes));
    }
}
```

12. Agregamos la clase **nuestras_ofertas.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso52;

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

public class nuestras_ofertas extends AppCompatActivity {

    ListView lvOfertas;
    String[][] ofertas-{{'HP Desktop','i5 8Gb','8','PEN 600.00'},
          {'HP Desktop Latitud','i3','6','PEN 350.00'},
          {'Enkope','i7 32Gb','9','PEN 3500.00'}};

    int[] imagenes-{R.drawable.ofe01,R.drawable.ofe02,R.drawable.ofe03};


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

        //Asociar el objeto al control ListView
        lvOfertas - (ListView) findViewById(R.id.lvOfertas);
        //Asociar el adaptador al control ListView
        lvOfertas.setAdapter(new Adaptador(this,ofertas,imagenes));
    }
}
```

13. En la clase **MainActivity.java**, colocamos el siguiente código:

```
package com.torres.manuel.pmcaso52;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.CardView;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    CardView cvProductos;
    CardView cvServicios;
    CardView cvOfertas;

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

        cvProductos = (CardView)findViewById(R.id.cvProductos);
        cvServicios = (CardView)findViewById(R.id.cvServicios);
        cvOfertas  = (CardView)findViewById(R.id.cvOfertas);

        cvProductos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
Intent visor=new Intent(getApplicationContext(),nuestros_productos.class);
                startActivity(visor);
            }
        });

        cvServicios.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
  Intent visor=new Intent(getApplicationContext(),nuestros_servicios.class);
                startActivity(visor);
            }
        });

        cvOfertas.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent visor=new
Intent(getApplicationContext(),nuestras_ofertas.class);
                startActivity(visor);
            }
        });
    }
}
```

14. En el archivo **styles.xml**, colocamos el siguiente código:

```
<resources>

    <!-- Base application theme. -->
    <style name='AppTheme' parent='Theme.AppCompat.Light.NoActionBar'>
        <!-- Customize your theme here. -->
        <item name='colorPrimary'>@color/colorPrimary</item>
        <item name='colorPrimaryDark'>@color/colorPrimaryDark</item>
```

```
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

15. El explorador del proyecto debe mostrarse de la siguiente manera:



CASO DESARROLLADO 03: **manejo de TabLayout**

Implementemos una aplicación móvil que permita mostrar información referente a diferentes mascotas, por ejemplo, la información de razas de perros, gatos y otros animales como loros. Esto debe realizarse como parte del servicio de información de una veterinaria a sus usuarios. El manejo de la aplicación se muestra en las siguientes imágenes:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 50):

Tabla 50. Especificación de valores para el proyecto

Create Android Project	Application Name	: pmCaso53
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_05
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar	: Empty Activity
	Activity Name	: MainActivity
	Layout Name	: activity_main

3. Ahora, debemos copiar las imágenes que usaremos en el proyecto en la carpeta **drawable**. Esto se realizará pegando directamente en dicha carpeta, de tal forma que se muestre de la siguiente manera:



4. Realizamos el diseño del prototipo colocando el siguiente código en el archivo **activity_main.xml**:



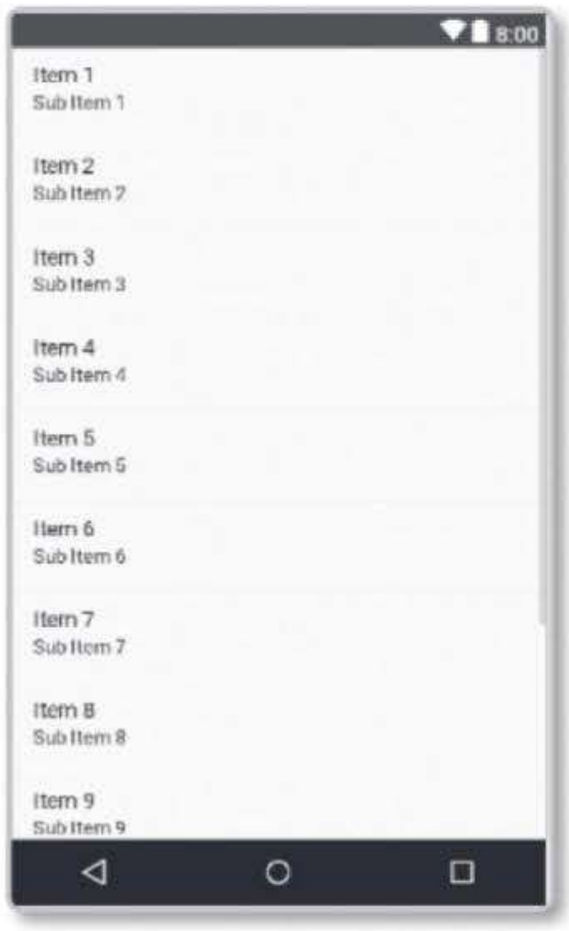

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/banner"
        android:minHeight="?attr/actionBarSize" />

    <android.support.design.widget.TabLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabGravity="fill"
        app:tabMode="fixed"
        android:id="@+id/tabLayout_id"
        android:background="#0000FF"
        app:tabTextColor="#FFF" />

    <android.support.v4.view.ViewPager
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/viewpager_id"/>
</LinearLayout>
```

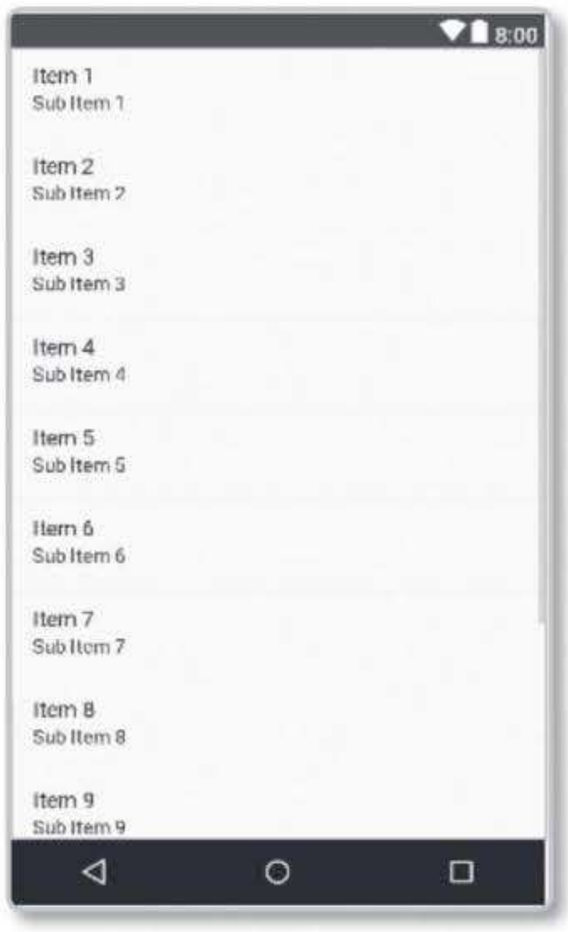
5. Agregamos al proyecto un fragment de tipo Fragment(Blank) y le asignamos el nombre Perros_fragment.xml. Esto nos permitirá definir el aspecto que tendrá el historial de perros en el ListView. Al crear el fragment, desactivamos los checks > "Include fragment factory methods" e "include interface callbacks", y luego agregamos un ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<FrameLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.PerrosFragment'>

    <!-- TODO: Update blank fragment layout -->
    <ListView
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:id='@+id/lvPerros'
    />
</FrameLayout>
```

6. Agregamos al proyecto un fragment de tipo Fragment(Blank) y le asignamos el nombre gatos_fragment.xml. Esto nos permitirá definir el aspecto que tendrá el historial de perros en el ListView. Al crear el fragment, desactivamos los checks > "Include fragment factory methods" e "include interface callbacks" y luego agregamos un ListView.



```
<?xml version='1.0' encoding='utf-8'?>
<FrameLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.gatosFragment'>

    <!-- TODO: Update blank fragment layout -->
    <ListView
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:id='@+id/lvGatos'
        />
</FrameLayout>
```

7. Agregamos al proyecto un fragment de tipo Fragment(Blank) y le asignamos el nombre otros_fragment.xml. Esto nos permitirá definir el aspecto que tendrá el historial de perros en el ListView. Al crear el fragment, desactivamos los checks > "Include fragment factory methods" e "include interface callbacks" y luego agregamos un ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<FrameLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.otrosFragment'>

    <!-- TODO: Update blank fragment layout -->
    <ListView
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:id='@+id/lvOtros'
        />
</FrameLayout>
```

8. Agregamos al proyecto un activity de tipo Empty Activity y le asignamos el nombre elemento_lista.xml. Esto nos permitirá mostrar un texto descriptivo de la mascota en el control ListView.




```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    tools:context='.elemento_lista'>

    <ImageView
        android:id='@+id/ivImagen'
        android:layout_width='130dp'
        android:layout_height='125dp'
        android:scaleType='fitXY'
        android:layout_margin='10dp'
        app:srcCompat='@drawable/boxer' />

    <LinearLayout
        android:layout_width='match_parent'
        android:layout_height='match_parent'
        android:orientation='vertical'>

        <TextView
            android:id='@+id/tvTitulo'
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:textStyle='bold'
            android:text='TextView'
            android:textSize='20dp' />
        <TextView
            android:id='@+id/tvDescripcion'
            android:layout_width='wrap_content'
            android:layout_height='wrap_content'
            android:text='TextView' />
    </LinearLayout>
</LinearLayout>
```

9. Agregamos la clase **Adaptador.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
```

```
import android.widget.ImageView;
import android.widget.TextView;

public class Adaptador extends BaseAdapter {

    private static LayoutInflater inflater=null;

    Context context;
    String[][] informacion;
    int[] imagenes;

    //Metodo Constructor
     public Adaptador(Context context, String[][] informacion, int[]
imagenes) {
        this.context = context;
        this.informacion = informacion;
        this.imagenes = imagenes;
        inflater = (LayoutInflater)
                context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //Asociar la vista al activity_elemento_lista.xml
        final View vista= inflater.inflate(R.layout.activity_elemento_
lista,null);

        //Obtener todos los controles
        ImageView ivImagen= (ImageView)vista.findViewById(R.id.ivImagen);
        TextView tvTitulo= (TextView) vista.findViewById(R.id.tvTitulo);
          TextView tvDescripcion=(TextView) vista.findViewById(R.
id.tvDescripcion);

        //Asignar informacion a los controles por medio del arreglo
        tvTitulo.setText(informacion[position][0]);
        tvDescripcion.setText(informacion[position][1]);
        ivImagen.setImageResource(imagenes[position]);
        return vista;
    }

    //Metodo que devuelve la cantidad de imagenes
    @Override
    public int getCount() {
        return imagenes.length;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }
    @Override
    public long getItemId(int position) {
        return 0;
    }
}
```

10. En la clase **perros_fragment.java**, agregamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.AppCompatActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

/**
 * A simple {@link Fragment} subclass.
 */
public class PerrosFragment extends Fragment {

    //Declarar el objeto ListView
    ListView lvImagen;

    //Definicion de un arreglo de platos tipicos
    String[][] datos={
            {'Boxer','El perro bóxer (Deutscher Boxer), también conocido
como \'bóxer alemán\' o simplemente \'bóxer\' es una de las razas caninas
de tipo moloso más populares del mundo y nace del cruce entre un Brabant
bullenbeisser y un Bulldog, razas extintas actualmente. Debemos saber que
la raza bóxer apareció por primera.'},
            {'Cavalier','La raza de perro Cavalier King Charles spaniel
es conocida por su aparición múltiples peliculas, asi mismo, también se
popularizó gracias a las celebridades que lo eligieron como perro de
compañia, tales como Coco Chanel, Oscar Wilde o Frank Sinatra. Además,
esta raza es muy apreciada por su aspecto.'},
            {'Cocker Spaniel','Con un porte regio y elegante, gracias al
que luce un aspecto realmente señorial, combinado con un carácter sumamente
dulce y equilibrado, este animal conseguirá ganarse nuestro corazón con
esa mirada tan tierna en milésimas de segundo. Estos canes resultan los
compañeros ideales para todos aquellos dispuestos.'},
            {'FoxHound Americano','El foxhound americano es un perro de caza
desarrollado en Estados Unidos. Desciende del foxhound inglés, uno de los
perros de tipo sabueso más populares en Reino Unido. Podemos diferenciarlos
por las extremidades, notablemente más largas y finas en los individuos de
origen estadounidense.'},
            {'Lobero Irlandés','El lobero irlandés, también conocido como
lebrel irlandés o cazador de lobos irlandés, es una de las razas caninas más
queridas y apreciadas de Irlanda. Su historia es antigua y remota, puesto
que ya se menciona por Julio César en los comentarios sobre la guerra de
las Galias. Por entonces era valorada.'},
            {'Mastin','El mastín del Pirineo es un perro que alcanza un
tamaño considerable y, debido a ello, puede resultar imponente. Sin embargo,
debemos saber que en realidad se trata de un perro protector, afable y
tierno con los suyos, por lo que es sin lugar a dudas una raza especialmente
querida para quienes han tenido.'},
    };

    //Definicion del arreglo de imagenes
    int[] imagenes={R.drawable.boxer,R.drawable.cavalier,R.drawable.cocker_
spaniel,R.drawable.foxhound_americano,
                    R.drawable.lobero_ingles,R.drawable.mastin};
    public PerrosFragment() {

    }
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v=inflater.inflate(R.layout.fragment_perros,container,false);
        //Asociar el objeto al control ListView
        lvImagen = (ListView) v.findViewById(R.id.lvPerros);
        //Asociar el adaptador al control ListView
        lvImagen.setAdapter(new Adaptador(getActivity(),datos,imagenes));
        return v;
    }
}
```

11. En la clase **gatos_fragment.java**, agregamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;
/**
 * A simple {@link Fragment} subclass.
 */
public class gatosFragment extends Fragment {

    //Declarar el objeto ListView
    ListView lvImagen;

    //Definicion de un arreglo de platos tipicos
    String[][] datos={
            {'Gato sphynx o esfinge','El gato sphynx o gato esfinge es un
gato realmente único en el mundo. Fue el primero en ser aceptado como raza
sin pelo o sin pelaje aparente y lo que es seguro, es que crean agrado y
desagrado a la vez por parte de la sociedad humana. Muchos profesionales
sugieren que proviene de la raza devon rex.'},
            {'Savannah','Con una apariencia exótica y única, el gato savannah
recuerda a un auténtico leopardo en miniatura, ¿cierto? Pero que eso no
nos engañe, hablamos de un felino doméstico que se adapta perfectamente a
la convivencia dentro del hogar. Además, hablamos de un gato especialmente
activo, sociable y cariñoso.'},
            {'Gato nebelung','De un precioso y llamativo gris perla, con un
manto largo y sedoso, así lucen los gatos de la raza nebelung, unos rasgos
heredados de los gatos azules rusos, en el caso del color, y de los gatos
longhair americanos, en cuanto a la suavidad y largo del pelaje. Pero no
solo su imagen es adorable.'},
            {'Gato siberiano','De abundante manto y mirada penetrante,
el gato siberiano se ha convertido en una de las razas felinas más
populares y apreciadas en todo el mundo. Su temperamento equilibrado y sus
características físicas, lo han convertido, además, en uno de los compañeros
ideales para personas de todo tipo.'},
            {'Gato montés','En ExpertoAnimal descubriremos los detalles
sobre una raza bastante desconocida y con la que debemos tomar ciertas
precauciones si pretendemos adoptar e incluir en nuestra familia a uno de
estos felinos. Aunque haya quien los tenga como mascota, estos gatos son
salvajes y están catalogados como especie.'}};

    //Definicion del arreglo de imagenes
    int[] imagenes={R.drawable.gato_esfinge,R.drawable.savannah,R.drawable.
gato_nebelung,R.drawable.gato_siberiano,
```

```
                    R.drawable.gato_montes};
        public gatosFragment() {
            // Required empty public constructor
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
            View v=inflater.inflate(R.layout.fragment_gatos,container,false);
            //Asociar el objeto al control ListView
            lvImagen = (ListView) v.findViewById(R.id.lvGatos);
            //Asociar el adaptador al control ListView
            lvImagen.setAdapter(new Adaptador(getActivity(),datos,imagenes));
            return v;
        }
    }
```

12. En la clase **otros_fragment.java**, agregamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;
/**
 * A simple {@link Fragment} subclass.
 */
public class otrosFragment extends Fragment {

    //Declarar el objeto ListView
    ListView lvImagen;

    //Definicion de un arreglo de platos tipicos
    String[][] datos={
            {'Guacamayo Aliverde','El guacamayo rojo o guacamayo aliverde
es una especie de ave de la familia de los loros, del orden Psittaciformes.
Son grandes guacamayos que atraen la atención por su vivido y espectacular
colorido, y su llamado estridente con el que se comunican durante el vuelo.
Los ejemplares jóvenes se parecen a los adultos, pero tienen la cola más
corta.'},
            {'Guacamayo Azul y Amarillo','El guacamayo azul y amarillo es un
guacamayo netamente sudamericano: su área de distribución comprende desde
Panamá hasta el norte de Argentina, pasando por toda la cuenca amazónica.
En cautividad se reproducen fácilmente; en libertad viven en grupos de 25-30
ejemplares. Es conocido como guacamayo azul y amarillo, guacamayo azul y
dorado o guacamayo guardiazul. Este loro es el representante inconfundible
de los guacamayos, loros con poderosos picos y colas muy largas.'},
            {'Perico Frentirroj','El perico frentirrojo es una especie de
ave perteneciente a la familia de los loros. Es originario de Centroamérica,
específicamente entre las regiones del sur de Nicaragua, el este de Costa
Rica y el oeste de Panamá.'}};

    //Definicion del arreglo de imagenes
    int[] imagenes={R.drawable.guacamayo_aliverde,R.drawable.guacamayo_
azul,R.drawable.perico_frentirojo};
    public otrosFragment() {
        // Required empty public constructor
```

```
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v=inflater.inflate(R.layout.fragment_otros,container,false);
        //Asociar el objeto al control ListView
        lvImagen = (ListView) v.findViewById(R.id.lvOtros);
        //Asociar el adaptador al control ListView
        lvImagen.setAdapter(new Adaptador(getActivity(),datos,imagenes));
        return v;
    }
}
```

13. Agregamos la clase **ViewPagerAdapter.java** al proyecto y colocamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

import java.util.ArrayList;
import java.util.List;

public class ViewPagerAdapter extends FragmentPagerAdapter {

    private final List<Fragment> fragmentList = new ArrayList<>();
    private final List<String> fragmentListTitulos = new ArrayList<>();

    public ViewPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int i) {
        return fragmentList.get(i);
    }

    @Override
    public int getCount() {
        return fragmentListTitulos.size();
    }

    @Nullable
    @Override
    public CharSequence getPageTitle(int position) {
        return fragmentListTitulos.get(position);
    }

    public void addFragment(Fragment fragment, String titulo){
        fragmentList.add(fragment);
        fragmentListTitulos.add(titulo);
    }

}
```

14. En la clase **MainActivity.java**, colocamos el siguiente código:

```
package com.torres.manuel.pmcaso53;

import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    TabLayout tabLayout;
    ViewPager viewPager;

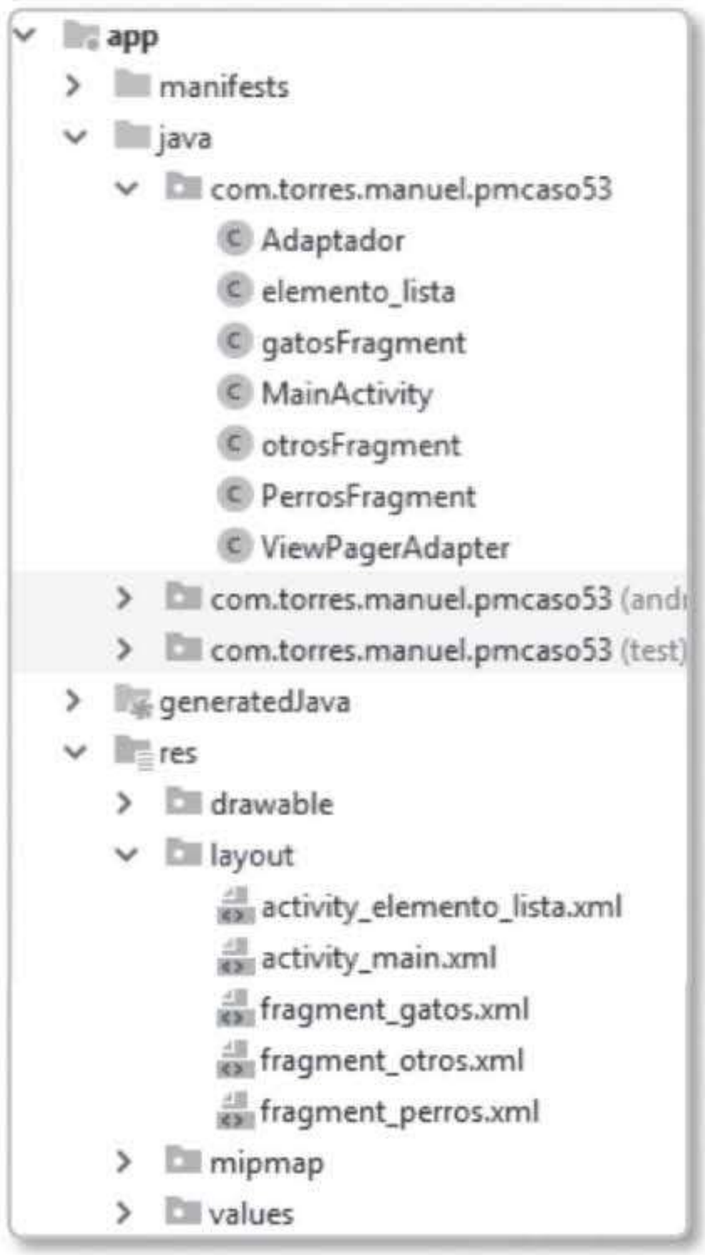
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tabLayout=(TabLayout)findViewById(R.id.tabLayout_id);
        viewPager=(ViewPager)findViewById(R.id.viewpager_id);
    ViewPagerAdapter adapter=new ViewPagerAdapter(getSupportFragmentManager());
        adapter.addFragment(new PerrosFragment(),'PERROS');
        adapter.addFragment(new gatosFragment(),'GATOS');
        adapter.addFragment(new otrosFragment(),'OTROS');
        viewPager.setAdapter(adapter);
        tabLayout.setupWithViewPager(viewPager);
    }
}
```

15. En el archivo **styles.xml**, colocamos el siguiente código:

```
<resources>
    <!-- Base application theme. -->
    <style name='AppTheme' parent='Theme.AppCompat.Light.NoActionBar'>
        <!-- Customize your theme here. -->
        <item name='colorPrimary'>@color/colorPrimary</item>
        <item name='colorPrimaryDark'>@color/colorPrimaryDark</item>
        <item name='colorAccent'>@color/colorAccent</item>
    </style>
</resources>
```

16. El explorador del proyecto debe mostrarse de la siguiente manera:



CAPÍTULO

6



Manejo de archivos

6.1 Persistencia de datos: Files

Las aplicaciones móviles no solo son implementadas para mostrar cierta información, sino también para interactuar con el usuario para la manipulación de datos. Entonces, existen aplicaciones que guardan información en archivos dentro del dispositivo; por ejemplo, guardar datos de correos y claves o información de preferencias o también podrían ser grandes cantidades de información, las cuales se guardan en base de datos. Para esto, Android cuenta con diferentes métodos de almacenamiento:

- Preferencias compartidas: Almacenamiento de datos primitivos privados en pares clave-valor.
- Almacenamiento interno: Almacenamiento de datos privados en la memoria del dispositivo.
- Almacenamiento externo: Almacenamiento de datos públicos en el almacenamiento externo compartido.
- Bases de datos SQLite: Almacenamiento de datos estructurados en una base de datos privada.
- Conexión de red: Almacenamiento de datos en la web mediante un servidor de red.

6.1.1 Preferencias compartidas

También llamadas Shared Preferences. Gracias a estas se puede almacenar y/o recuperar información con el formato de pares clave-valor. Dicha información puede ser del tipo números y textos. Los modos de acceso posibles son los que siguen:

- MODE_PRIVATE: Se limita a que solo nuestra aplicación podrá tener acceso a estas preferencias.
- MODE_WORLD_READABLE: Habilita la posibilidad de acceso de todas las aplicaciones, pero la modificación solo lo puede realizar una de ellas.
- MODE_WORLD_WRITEABLE: Habilita que todas las aplicaciones puedan leer y modificar las preferencias.

Veamos algunos script Java que permiten especificar la configuración de las preferencias:

```
SharedPreferences miPreferencia =
                    getSharedPreferences("Especificacion de preferencia",Context.MODE_PRIVATE);
```

6.1.2 Almacenamiento interno

Implica almacenar cierto tipo de información en la memoria del dispositivo. Además, consiste en guardar y recuperar la información en archivos normalmente de tipo texto. Hay que tener en cuenta que la prioridad de registro de información lo tiene la memoria interna del dispositivo, la cual puede presentar ciertas limitaciones, pero se ajusta a las necesidades del usuario.

La clase que permite registrar información en archivos es el OutputStreamWriter y, en el caso de apertura de información, usamos la clase openFileOutput. Estos cuentan con nombre del archivo y modo de acceso (lectura o escritura). Asimismo, los modos de acceso se presentan de la siguiente manera:

- MODE_PRIVATE: Accesible solo por el dispositivo que lo crea. Si el archivo es nuevo, se creará, y si ya existe, solo se actualizará.

- MODE_APPEND: Permite agregar un contenido a lo almacenado dentro de un determinado archivo.
- MODE_WORLD_READABLE: Habilita que todas las aplicaciones puedan ver el contenido, pero no modificarlo.
- MODE_WORLD_WRITEABLE: Habilita que ciertas aplicaciones puedan editar el contenido del archivo.

Veamos algunos script Java que permiten especificar la configuración de las preferencias:

```
//Creando el archivo informacion.txt
OutputStreamWriter osw=new OutputStreamWriter(
                    openFileOutput('informacion.txt', Context.MODE_PRIVATE));

//Enviar informacion al archivo informacion.txt
outputStreamWriter.write('Contenido del archivo');
outputStreamWriter.close();

//Obtener datos del archivo informacion.txt
BufferedReader br = new BufferedReader(new InputStreamReader(
                        openFileInput('informacion.txt')));

String cadena = br.readLine();
Log.i('ddd',texto);
br.close();
```

6.1.3 Almacenar archivos en memoria externa

Una de las características de las aplicaciones Android es poder registrar información o leerla desde la memoria externa o también llamada memoria SD. Para esto, debemos especificar su uso dentro del archivo manifest, tal como se muestra en el siguiente script:

```
<uses-permission android:name='android.permission.READ_EXTERNAL_STORAGE'
/>
<uses-permission android:name='android.permission.WRITE_EXTERNAL_STORAGE'
/>
```

Antes de ejecutar operaciones haciendo uso de la memoria externa, debemos considerar los estados que presenta el método Environment.getExternalStorageStatus():

- Environment.MEDIA_MOUNTED: Indica que la memoria está disponible y "montada" en el dispositivo; en consecuencia, podemos acceder a esta.
- Environment.MEDIA_MOUNTED_READ_ONLY: Indica que la memoria está disponible y "montada" en el dispositivo, pero solo podemos leer en ella.
- Environment.MEDIA_UNMOUNTED o Environment.MEDIA_REMOVED: Indica que la memoria no está disponible y que, por lo tanto, no podemos ni leer ni escribir en esta.

6.2 Clase File

Es usado principalmente para la identificación y gestión de los archivos y carpetas dentro del sistema operativo Android. Sus características son heredadas de la clase File de Java. Mediante sus métodos, podemos leer y escribir sobre un determinado archivo, pero debemos tener en cuenta

su ubicación. Esta se puede dar mediante una ruta absoluta, la cual se brinda en el sistema de archivos de Android, y la ruta relativa, que es donde se encuentra la aplicación.

La clase presenta la siguiente forma:

File(Carpeta, Nombre del archivo)
Crea un archivo en una determinada carpeta y con un respectivo nombre, especificados como parámetros.

File(Carpeta)
Crea un archivo en una carpeta determinada, especificada como parámetro.

File(Uri)
Crea un archivo usando el parámetro URI (identificador de recurso uniforme), el cual identifica rápidamente un recurso como un documento, dirección de correo, etc.

En la Tabla 52, se muestran los principales métodos:

Tabla 52. Métodos de la clase File

canExecute	Descripción
canRead	Emite un valor booleano que permite evaluar si la aplicación Android puede ejecutar el archivo.
canWrite	Emite un valor booleano que permite evaluar si la aplicación Android puede leer el contenido de un archivo.
Delete	Emite un valor booleano que permite evaluar si la aplicación Android puede escribir dentro de un archivo.
exists	Emite un valor booleano que permite evaluar si la aplicación Android puede eliminar un archivo.
getName	Emite un valor booleano que permite evaluar si un archivo existe.
getPath	Devuelve el nombre del archivo o carpeta que representa a un determinado archivo.
isDirectory	Permite devolver la ruta de un determinado archivo.
isFile	Emite un valor booleano que indica si la clase File contiene una carpeta.
lenght	Emite un valor booleano que indica si la clase File contiene un archivo.
renameTo	Devuelve un valor decimal que indica el tamaño del archivo en bytes.
	Permite modificar el nombre del archivo.

6.3 Clase FileOutputStream

Clase que, mediante sus métodos, permite agregar datos a un archivo para mantenerlo registrado. Para crear y agregar datos a un archivo con la clase FileOutputStream, debemos seguir los siguientes pasos:

1. Apertura de un FileOutputStream: Aquí se especifica qué archivo se va a abrir para guardar la información. Adicionalmente, se debe especificar el modo de apertura:

```
FileOutputStream fos=openFileOutput(archivo, Context.MODE_PRIVATE);
```

2. Escritura en un FileOutputStream: Es el método que permite enviar la informacion al archivo. Esto se realiza con el método getBytes para poder obtener cada valor que se enviará al archivo.

```
fos.write(cadena.getBytes());
```

3. Cerrar el objeto de tipo FileOutputStream: Para finalizar el registro de información sobre el archivo, se debe cerrar este mediante el método close.

```
fos.close();
```

Por ejemplo, para crear un archivo llamado mitexto.txt en el dispositivo, podemos considerar el siguiente código:

```
try {
 String cadena='Desarrollo de Aplicaciones Android';
 FileOutputStream fos=openFileOutput('mitexto.txt', Context.MODE_PRIVATE);
 fos.write(cadena.getBytes());
 fos.close();
}catch(Exception e){
 e.printStackTrace();
}
```

6.4 Clase FileInputStream

Es usada normalmente para obtener información de un archivo. La forma de acceder se realiza de forma secuencial, es decir, byte a byte. Para crear y agregar datos a un archivo con la clase FileOutputStream, debemos seguir los siguientes pasos:

1. Apertura de un FileInputStream: Aquí se especifica qué archivo se va a abrir para leer su información:

```
FileInputStream fis=openFileInput(archivo);
```

2. Leer la información del FileInputStream: Se debe hacer uso de la clase Buffer para guardar temporalmente la información obtenida desde el archivo.

```
int tamaño=fis.available();
byte[] buffer=new byte[tamaño];
fis.read(buffer);
```

3. Cerrar el objeto de tipo FileOutputStream: Como se realizó en la creación del archivo, también al abrir, el archivo se debe cerrar.

```
fis.close();
```

Por ejemplo, para crear un archivo llamado mitexto.txt en el dispositivo, podemos considerar el siguiente código:

```
String archivo-'mitexto.txt':
String cadena-'';
try {
 FileInputStream fis-openFileInput(archivo);
 int tamaño-fis.available();
 byte[] buffer-new byte[tamaño];
 fis.read(buffer);
 fis.close();
 cadena-new String(buffer);
}catch(Exception e){
 e.printStackTrace();
}
etCadena.setText(cadena);
```

6.5 Leer XML Data

Android permite leer datos de un archivo XML a partir de la clase XmlPullParserFactory, la cual contiene métodos de acceso que tratan a los archivos XML.

Debemos iniciar por crear un objeto de la clase XmlPullParserFactory de la siguiente manera:

```
XmlPullParserFactory xpf-XmlPullParserfactory.newInstance();
XmlPullParser parser - parserfactory.newPullParser();
```

Para poder leer un archivo XML, este deberá encontrarse precargado en nuestra aplicación, y para hacerle referencia, debemos usar la siguiente línea de código:

```
InputStream is - getAssets().open('archivo.xml');
parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
parser.setInput(is, null);
```

Para poder recorrer por el contenido del archivo XML, debemos usar una estructura repetitiva hasta que sea el fin del archivo como lo implementa while o for, tal como se visualiza en el siguiente código:

```
while(eventType != XmlPullParser.END_DOCUMENT){
```

CASO DESARROLLADO 01: **manejo de la clase FileOutputStream**

Implementemos una aplicación móvil en Android que permita agregar y mostrar la información registrada en un archivo .txt. El prototipo de la aplicación debe mostrarse de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 53):

Tabla 53. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso61 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_06
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Modificamos el archivo activity_main.xml, de forma que el código se muestre de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <TextView
        android:id='@+id/textView2'
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="CADENA ORIGINAL" />

    <EditText
        android:id="@+id/etCadena"
        android:layout_width="382dp"
        android:layout_height="192dp"
        android:gravity="top"
        android:inputType="textMultiLine" />

    <Button
        android:id="@+id/btAbrir"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Abrir"  />

    <Button
        android:id="@+id/btGuardar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Guardar"  />
    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="CADENA REGISTRADA" />
    <TextView
        android:id="@+id/tvCadena"
        android:layout_width="match_parent"
        android:layout_height="219dp"
        android:gravity="top"
        />
</LinearLayout>
```

4. Modificamos el archivo **MainActivity.java**, de forma que el código se muestre de la siguiente manera:

```
package com.example.toshibapro.pmcaso61;

import android.content.Context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.FileInputStream;
import java.io.FileOutputStream;

public class MainActivity extends AppCompatActivity {

    EditText etCadena;
    TextView tvCadena;
    Button btAbrir, btGuardar;

    String archivo= "cadena.txt";
```

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

        etCadena = (EditText) findViewById(R.id.etCadena);
        tvCadena = (TextView) findViewById(R.id.tvCadena);

        btAbrir = (Button) findViewById(R.id.btAbrir);
        btAbrir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                tvCadena.setText(leer(archivo));
            }
        });

        btGuardar = (Button) findViewById(R.id.btGuardar);
        btGuardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                guardar(archivo,etCadena.getText().toString());
            }
        });
    }

    //Metodo para grabar
    public void guardar(String archivo, String cadena){
        try {
         FileOutputStream fos=openFileOutput(archivo, Context.MODE_PRIVATE);
          fos.write(cadena.getBytes());
          fos.close();
          Toast.makeText(this,'Grabado correctamente',Toast.LENGTH_SHORT).
show();

        }catch(Exception e){
            e.printStackTrace();
             Toast.makeText(this,'Error al grabar',Toast.LENGTH_SHORT).
show();
        }
    }

    public String leer(String archivo){
        String cadena='';
        try {
            FileInputStream fis=openFileInput(archivo);
            int tamaño=fis.available();
            byte[] buffer=new byte[tamaño];
            fis.read(buffer);
            fis.close();
            cadena=new String(buffer);
        }catch(Exception e){
            e.printStackTrace();
       Toast.makeText(this,'Error al leer el archivo',Toast.LENGTH_SHORT).
show();
        }
        return cadena;
    }
}
```

CASO DESARROLLADO 02: **Manejo de la clase InputStream**

Implementemos una aplicación móvil en Android que permita mostrar el contenido de un archivo de texto usando la clase InputStream. El prototipo de la imagen debe mostrarse de la siguiente manera:



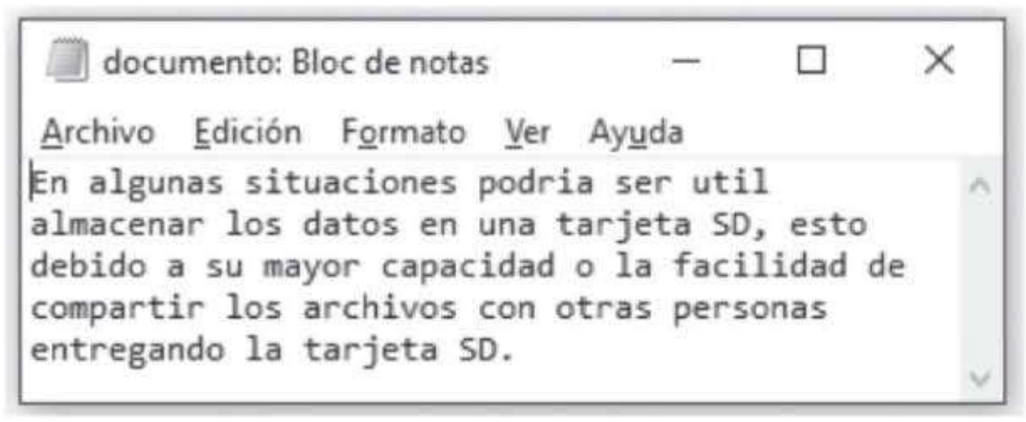
Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 54):

Tabla 54. Especificación de valores para el proyecto

Create Android Project	Application Name Company Domain Project Location	: pmCaso62 : Modifique por un nombre adecuado : C:\Libro Android\Capitulo_06
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar Activity Name Layout Name	: Empty Activity : MainActivity : activity_main

3. Iniciamos creando la carpeta **assets**, la cual contendrá el archivo .txt que leerá la aplicación. Para esto, debemos presionar clic derecho sobre la carpeta **"app" > new > Android Resource Directory** y asignarle el nombre assets. Luego, desde el explorador de archivos de Windows, movemos la carpeta a la dirección **> ...\pmCaso62\app\src\main\assets**.
4. Nos ubicamos dentro de la carpeta **assets** y agregamos un archivo de tipo .txt. Luego, rellenamos con información que será obtenida por la aplicación:



5. El explorador de aplicaciones debe mostrarse de la siguiente manera:



6. Realizamos modificaciones en el **archivo activity_main.xml**, tal como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <EditText
        android:id='@+id/etCadena'
        android:layout_width='match_parent'
        android:layout_height='304dp'
        android:ems='10'
        android:gravity='top'
        android:inputType='textMultiLine' />

    <Button
        android:id='@+id/btMostrar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='MOSTRAR' />

</LinearLayout>
```

7. Realizamos modificaciones en el archivo MainActivity.java, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso62;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import java.io.InputStream;

public class MainActivity extends AppCompatActivity {

    EditText etCadena;
```

```
    Button btMostrar;

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

        etCadena = (EditText) findViewById(R.id.etCadena);
        btMostrar = (Button)findViewById(R.id.btMostrar);
        btMostrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String cadena="";
                try {
                    InputStream is= getAssets().open("documento.txt");
                    int tamaño=is.available();
                    byte[] buffer=new byte[tamaño];
                    is.read(buffer);
                    is.close();
                    cadena=new String(buffer);

                }catch(Exception e){
                    e.printStackTrace();
                }
                etCadena.setText(cadena);
            }
        });
    }
}
```

CASO DESARROLLADO 03: **leer XML**

Implementemos una aplicación móvil que permita obtener la información de los productos previamente registrados en un archivo .xml, el cual se muestra de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<productos>
    <producto>
        <codigo>P001</codigo>
        <descripcion>LAVADORA</descripcion>
        <precio>1500</precio>
        <stock>100</stock>
    </producto>
    <producto>
        <codigo>P002</codigo>
        <descripcion>TELEVISOR</descripcion>
        <precio>2500</precio>
        <stock>50</stock>
    </producto>
    <producto>
        <codigo>P003</codigo>
        <descripcion>TOSTADORA</descripcion>
        <precio>150</precio>
        <stock>10</stock>
    </producto>
</productos>
```

El diseño de la aplicación debe mostrarse del siguiente modo:



Debemos tener en cuenta que dicho archivo debe encontrarse pregrabado en el archivo **productos.xml** y ubicado en la carpeta > ...**\pmCaso63\app\src\main\assets**.

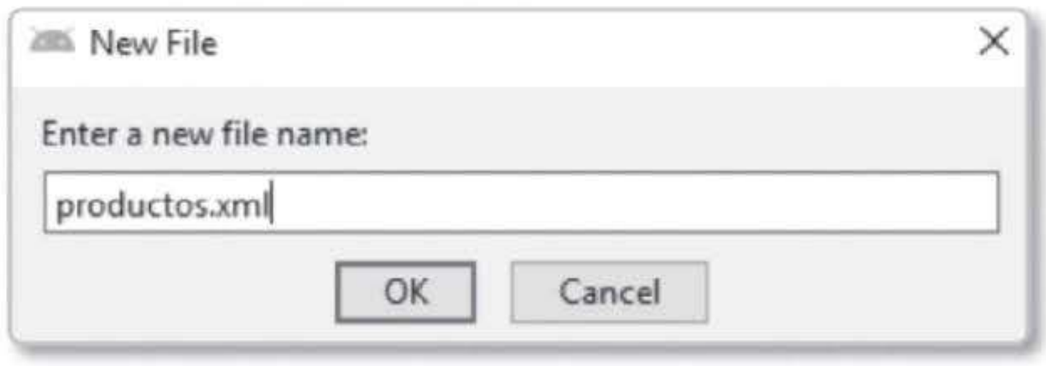
Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 55):

Tabla 55. Especificación de valores para el proyecto

Create Android Project	Application Name : pmCaso63 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_06
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Iniciamos creando la carpeta **assets**, la cual contendrá el archivo .xml que leerá la aplicación. Para esto, debemos presionar clic derecho sobre la carpeta **"app" > new > Android Resource Directory** y asignarle el nombre assets. Luego, desde el explorador de archivos de Windows, movemos la carpeta a la dirección > ...**\pmCaso63\app\src\main\assets**.
4. En Android Studio, aparecerá la carpeta **assets** y allí añadimos el archivo **productos.xml**. Para ello, presionamos clic derecho sobre la carpeta **assets > new > Android Resource File** y colocamos el siguiente nombre:



5. Realizamos modificaciones en el archivo **activity_main.xml**, tal como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <TextView
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:textSize='20dp'
        android:textStyle='bold'
        android:gravity='center'
        android:text='LISTADO DE PRODUCTOS - XML'/>

    <EditText
        android:id='@+id/etCadena'
        android:layout_width='match_parent'
        android:layout_height='421dp'
        android:ems='10'
        android:inputType='textMultiLine' />

</LinearLayout>
```

6. Agregamos una clase al proyecto llamado Producto, y agregamos sus métodos get y set por cada atributo. Debemos considerar que los atributos dependen del contenido del archivo XML, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso63;

public class Producto {

    //Atributos
    private String codigo;
    private String descripcion;
    private double precio;
    private int stock;

    //Metodos Get y Set
    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
```

```
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
}
```

7. Realizamos modificaciones en el archivo **MainActivity.java**, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso63;

import android.content.Context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    EditText etCadena;

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

```
        etCadena = (EditText)findViewById(R.id.etCadena);
        //Permite mostrar los productos desde XML
        cargarXML();
    }

    //Metodo que permite cargar los productos desde XML
    private void cargarXML(){
        XmlPullParserFactory parserfactory;
        try {
            parserfactory = XmlPullParserFactory.newInstance();
            XmlPullParser parser = parserfactory.newPullParser();

            //Abrir el archivo productos.xml
            InputStream is = getAssets().open('productos.xml');
            parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
            parser.setInput(is, null);

            //Enviar los datos XML del archivo productos.xml
            processParsing(parser);
        }catch(XmlPullParserException e){
        }catch(IOException e){
        }
    }

    //Obtener los datos del XML y guardarlo en un arreglo
    private void processParsing(XmlPullParser parser)
                                throws IOException, XmlPullParserException {

        //Definir un arreglo de productos
        ArrayList<Producto> misProductos=new ArrayList<>();
        int eventType=parser.getEventType();
        Producto objP=null;

        //Recorrer por el XML para recuperar la informacion
        while(eventType != XmlPullParser.END_DOCUMENT){
            String eltName=null;
            switch (eventType){
                case XmlPullParser.START_TAG:
                    eltName = parser.getName();
                    if ('producto'.equals(eltName)){
                        objP =new Producto();
                        misProductos.add(objP);
                    }else if(objP != null){
                        if('codigo'.equals(eltName)){
                            objP.setCodigo(parser.nextText());
                        }else if('descripcion'.equals(eltName)){
                            objP.setDescripcion(parser.nextText());
                        }else if('precio'.equals(eltName)){
                        objP.setPrecio(Double.parseDouble(parser.nextText()));
                        }else if('stock'.equals(eltName)){
                        objP.setStock(Integer.parseInt(parser.nextText()));
                        }
                    }
                    break;
            }
            eventType = parser.next();
        }
        imprimirProductos(misProductos);
    }

    //Imprimir los productos obtenidos desde el arreglo
```

```
    private void imprimirProductos(ArrayList<Producto> aProductos ){
        StringBuilder builder=new StringBuilder();

        //Obtener los datos del arreglo
        for(Producto p : aProductos){
            builder.append(p.getCodigo()).append('\n').
                    append(p.getDescripcion()).append('\n').
                  append(String.format('%.2f',p.getPrecio())).append('\n').
                    append(p.getStock()).append('\n\n');
        }

        //Imprimir la informacion
        etCadena.setText(builder.toString());
    }
}
```

CAPÍTULO

7



Acceso a datos con SQLite

7.1 Fundamentos SQLite

Una aplicación móvil no solo se puede quedar con la información dentro del dispositivo; siempre habrá necesidad de compartir dicha información, lo cual depende del proceso de negocio en el cual nos enfoquemos. El uso de las bases de datos en Android se ha extendido conforme evoluciona la tecnología. SQLite es un gestor de base de datos para Android que permite administrar la información de un determinado proceso.



SQLite aparece a mediados del año 2000 por la necesidad de mantener información actualizada dentro de nuestras aplicaciones Android. Su creador, Richard Hip, liberó las versiones de SQLite bajo la licencia GPL, por lo que su código es totalmente administrable. Basa su soporte de consultas en el lenguaje SQL; por lo tanto, si usted está ambientado a consultas SQL o MySQL, entonces, no le será difícil entender las sentencias en SQLite.

Características del SQLite. Las que siguen son sus principales características:

- La base de datos completa se encuentra en un solo archivo y puede funcionar directamente desde la memoria del dispositivo, lo que facilita la rapidez en las consultas de sus archivos.
- No tiene dependencias externas, ya que es totalmente autocontenida.
- Es compatible con muchos lenguajes de programación, pues cuenta con un conjunto de librerías de soporte.
- Como todo gestor de base de datos, también soporta texto en formato UTF-8 y UTF-16, así como datos numéricos de 64 bits.
- Tiene un gran soporte a sentencias de SQL.
- Hay una comunidad que permite mantener el sistema SQLite; por eso, es considerado como código de dominio público y se encuentra documentado.

Plataformas de SQLite. SQLite cuenta con un soporte que permite embeber con muchas aplicaciones y principalmente sistemas operativos como los siguientes:

- Windows 95, 98, ME, 2000, XP y Vista
- Windows CE
- Linux
- OpenEmbedded
- PalmOS
- Symbian

7.2 Gestión de base de datos SQLite

Es un motor y gestor de base de datos SQL de código libre. Su configuración es sencilla y se caracteriza por almacenar información sin un servidor en particular. Su descarga se realiza de forma gratuita y contiene librerías para la mayoría de lenguajes de programación.

SQLite cumple con las características ACID (atomicidad, consistencia, aislamiento y durabilidad). Además, cuenta con una biblioteca especializada, la cual es llamada de forma dinámica a través de las diferentes aplicaciones que la usan.

En la Tabla 56, se muestra los tipos de datos con que cuenta SQLite.

Tabla 56. Tipos de datos SQLite

TIPO	DESCRIPCIÓN
Int	Tipo de datos numéricos sin fracciones decimales.
Varchar	Tipo de datos que almacena información textual, el cual también incluye valores numéricos y caracteres especiales.
Blob	Tipo de datos que almacena información de todo tipo, principalmente de gran tamaño, y que puede variar dinámicamente.
Real	Tipo de datos que almacena valores enteros y decimales.
Boolean	Tipo de datos que almacena valores true o false únicamente.
Date	Tipo de datos que almacena solo fechas.
DateTime	Tipos de datos que almacena fecha y hora.

7.3 Componente SQLiteOpenHelper

Es una clase abstracta que permite administrar las bases de datos de una aplicación. Esta clase contiene dos métodos principales en los que basa toda su funcionalidad: onCreate y onUpgrade. Por lo tanto, para usar dichos métodos, debemos sobreescribirlos. Por otro lado, debemos usar su método constructor para la gestión de la base de datos. Veamos su código inicial:

```
private class DBHelper extends SQLiteOpenHelper {

  public DBHelper(Context context, String name, CursorFactory factory, int version) {
     super(context, name, factory, version);
     // TODO Auto-generated constructor stub
  }
  @Override
  public void onCreate(SQLiteDatabase db) {
     // TODO Auto-generated method stub
  }
  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
     // TODO Auto-generated method stub
    }
}
```

7.4 Métodos de inserción, actualización y eliminación

Antes de iniciar con los procesos de inserción, actualización y eliminación, debemos entender cómo funcionan los métodos execSQL, los cuales permitirán crear una tabla dentro de una base de datos, y el método rawQuery, que permitirá visualizar los registros de una determinada consulta.

- execSQL: Ejecuta una sentencia SQL que pertenezca al lenguaje de definición de datos como Create, Alter, Drop.

```
String SQL='CREATE TABLE PRODUCTO (COD_PRO TEXT, DES_PRO TEXT PRE_PRO
REAL)';
db.execSQL(SQL);
```

- RawQuery(SentenciaSQL, ParametrosArreglo): Esta función permite recuperar un cursor (tabla) con el fin de cargar los datos en un adaptador para, luego, presentarlo en un contenedor tipo lista como un Listview.

```
String SQL='CREATE TABLE PRODUCTO (COD_PRO TEXT, DES_PRO TEXT PRE_PRO
REAL)';
db.execSQL(SQL);
```

Método para la inserción de registros:

```
SQLiteDatabase db= getWritableDatabase();
db.insert(<Nombre de la tabla>, null, <Datos a insertar>);
db.close();

Los datos a insertar se pueden dar de la siguiente forma:
ContentValues values=new ContentValues();
values.put(<Campo de la tabla>,<Valor a insertar>);
```

7.5 Clase ContentValues

Esta clase permite almacenar variables de tipo diccionario, es decir, un nombre y un valor de la variable. Se utiliza para pasar los datos a una tabla y ejecutar las sentencias de mantenimiento de datos en una base de datos de SQLite.

Formato para la definición de parámetros con ContentValues:

```
ContentValues parametros=new ContentValues();
parametros.put(<Campo>,<Valor>);
```

Por ejemplo, si necesitamos enviar información de un registro de producto, podríamos usar el siguiente código:

```
ContentValues values=new ContentValues();
values.put(cod_pro,'P001');
values.put(des_pro,'Lavadora');
values.put(pre_pro,'2500');
```

7.6 Cursores y contenedores de valores en SQLite

Representa un objeto de tipo tabla, el cual permite almacenar información resultante de una consulta. Los valores se enviarán a un cursor con la finalidad de recuperarlos en un contenedor de registros, por ejemplo, un ListView.

Formato para el uso de cursores:

```
Cursor nombreCursor = <BaseDatos>.rawQuery('<Consulta>', <Argumentos>);
```

Asimismo, se considera los siguientes métodos:

- getColumnCount(): Permite saber el número de columnas que tiene la tabla consultada.
- close(): Permite cerrar el cursor que se encuentra abierto.
- getColumnName(columnIndex): Permite leer el nombre de una columna a partir del índice.
- getCount(): Permite leer el número de registros existentes en el cursor.
- getDouble(columnIndex): Permite leer el valor de un campo número con decimales de tipo doublé.
- getString(columnIndex): Permite leer el valor de un campo de tipo cadena varchar, text nvarchar.
- getInt(columnIndex): Permite leer el valor de un campo de tipo entero.
- getFloat(columnIndex): Permite leer el valor de un campo de tipo número con decimales float.

7.7 Modelo Vista Controlador

En las aplicaciones de escritorio y web, se habla mucho de la programación en capas, pues mejora el rendimiento de una aplicación. En ese sentido, se expone, en la Tabla 57, el patrón MVC (Modelo Vista Controlador).

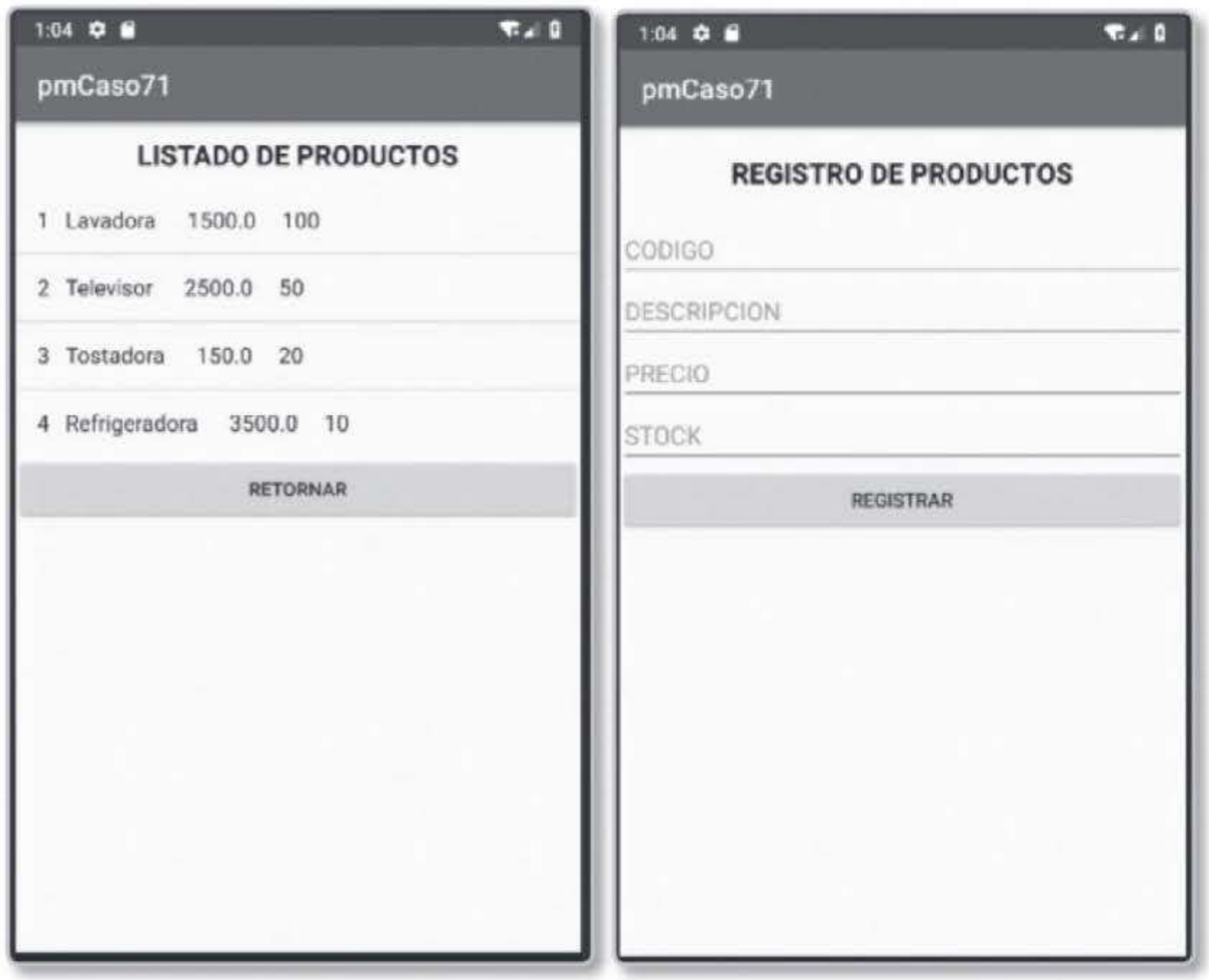
Tabla 57. Definición MVC

PATRÓN	DESCRIPCIÓN
Modelo	En esta capa, se define los métodos que permiten gestionar una tabla de una base de datos; es decir, listado, registro, actualización o eliminación. También, es considerada como la capa de la lógica de negocio, pues no tiene ningún elemento visual del usuario, solo código.
Vista	En esta capa, el usuario podrá interactuar con la aplicación ingresando información que le solicite la aplicación o seleccionando procesos que necesite. Esta capa contiene todas las actividades de una aplicación móvil.
Controlador	En esta capa, se gestiona las solicitudes que se puede hacer desde la vista hacia el modelo. Es decir, si en la vista se solicita un listado de productos, esta capa buscará dicho método en el modelo, devolverá la respuesta a la vista y finalizará la gestión.

Por otro lado, dentro de cada capa, se crean clases que realizan los procesos de negocio de una aplicación. La idea principal es mantener ordenadas las clases, pues una aplicación móvil puede desarrollarse solamente con clases, pero llegaremos, en algún momento, a tener muchas clases, lo cual generará desorden en la aplicación. Con MVC, podemos colocar las clases en sus respectivas capas.

CASO DESARROLLADO 01: **registro y listado de productos**

Implementemos una aplicación móvil en Android que permita registrar datos de productos como el código, descripción del producto, precio y *stock*. Este se almacenará en una base de datos llamada Ventas. Luego, desde otra actividad, debemos mostrar la lista de dichos productos. El prototipo del registro y listado de productos se muestra de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 58):

Tabla 58. Especificación de valores del proyecto

Create Android Project	Application Name : pmCaso71 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_07
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos una clase llamada Producto al proyecto. Luego, declaramos los atributos de la clase de tal forma que cada atributo sea par del campo de la tabla Producto, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso71;

public class Producto {
    private int codigo;
    private String descripcion;
    private double precio;
    private int stock;

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
}
```

4. Agregamos una clase llamada DAOProducto al proyecto, en la cual se definirán todos los métodos que serán necesarios para el listado y registro de productos. Además, definimos el proceso de creación de la base de datos Ventas y su tabla Producto, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso71;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import java.util.ArrayList;

public class DAOProducto extends SQLiteOpenHelper {

    //Definicion de la base de datos VENTAS y la tabla Producto
```

```
    private static final String base='VENTAS';
    private static final int version=1;
    private static final String tabla='PRODUCTO';

    //Definicion de las columnas
    private static final String cod_pro='COD_PRO';
    private static final String des_pro='DES_PRO';
    private static final String pre_pro='PRE_PRO';
    private static final String sto_pro='STO_PRO';

    public DAOProducto(Context context) {
        super(context, base, null, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       String SQL='CREATE TABLE '+tabla+' ('+cod_pro+' text PRIMARY KEY,'
                +des_pro+' text,'+pre_pro+' text,'
                +sto_pro+' text)';
        db.execSQL(SQL);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL('DROP TABLE IF EXISTS '+tabla);
        onCreate(db);
    }

    //Metodo para registrar un producto
    public void nuevoProducto(Producto objP) {
        ContentValues values=new ContentValues();
        values.put(cod_pro,objP.getCodigo());
        values.put(des_pro,objP.getDescripcion());
        values.put(pre_pro,objP.getPrecio());
        values.put(sto_pro,objP.getStock());
        SQLiteDatabase db= getWritableDatabase();
        db.insert(tabla, null, values);
        db.close();
    }

    //Metodo para listar los productos
    public ArrayList listadoProductos() {
        ArrayList listaProductos=new ArrayList();
        SQLiteDatabase db=getWritableDatabase();

        String SQL='SELECT * FROM '+tabla;
        Cursor c=db.rawQuery(SQL,null);

        if(c.moveToFirst()) {
            do {
              listaProductos.add(c.getString(0)+'  '+c.getString(1)+'    '
                        +c.getString(2)+'     '+c.getString(3));
            }while(c.moveToNext());
        }
        return listaProductos;
    }
}
```

5. Agregamos un blank activity al proyecto y lo llamamos activity_registro.xml. Luego, diseñamos el prototipo de registro de productos, tal como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="20dp"
        android:textStyle="bold"
        android:textColor="#000"
        android:gravity="center"
        android:text="REGISTRO DE PRODUCTOS" />

    <EditText
        android:id="@+id/etCodigo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="CODIGO"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/etDescripcion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="DESCRIPCION"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/etPrecio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="PRECIO"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/etStock"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="STOCK"
        android:inputType="textPersonName" />

    <Button
        android:id="@+id/btRegistrar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="REGISTRAR" />
</LinearLayout>
```

6. Modificamos el archivo **MainActivity.java** para agregar un nuevo producto a la base de datos Ventas:

```
package com.example.toshibapro.pmcaso71;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
                                    implements View.OnClickListener {

    EditText etCodigo,etDescripcion,etPrecio,etStock;
    Button btGrabar;
    DAOProducto objDAO;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        etCodigo = (EditText)findViewById(R.id.etCodigo);
        etDescripcion= (EditText)findViewById(R.id.etDescripcion);
        etPrecio = (EditText)findViewById(R.id.etPrecio);
        etStock = (EditText)findViewById(R.id.etStock);
        btGrabar = (Button) findViewById(R.id.btRegistrar);
        btGrabar.setOnClickListener(this);

        objDAO = new DAOProducto(this);
    }

    //Metodo OnClick
    @Override
    public void onClick(View v){
        Producto objP = new Producto();
        objP.setCodigo(Integer.parseInt(etCodigo.getText().toString()));
        objP.setDescripcion(etDescripcion.getText().toString());
       objP.setPrecio(Double.parseDouble(etPrecio.getText().toString()));
        objP.setStock(Integer.parseInt(etStock.getText().toString()));

        objDAO.nuevoProducto(objP);
       Toast.makeText(this,'Producto registrado Ok',Toast.LENGTH_SHORT).
show();
        Intent intent= new Intent(this,listado_productos.class);
        startActivity(intent);
    }
}
```

7. Agregamos un blank activity al proyecto y lo nombramos activity_listado_productos.xml. Luego, diseñamos el prototipo de listado de productos, tal como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
```

```
        android:orientation='vertical'
        tools:context='.listado_productos'>

        <TextView
            android:id='@+id/textView2'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:gravity='center'
            android:textColor='#000'
            android:textStyle='bold'
            android:textSize='20dp'
            android:layout_marginBottom='10dp'
            android:layout_marginTop='10dp'
            android:text='LISTADO DE PRODUCTOS' />

        <ListView
            android:id='@+id/lvProductos'
            android:layout_width='match_parent'
            android:layout_height='wrap_content' />

        <Button
            android:id='@+id/btRetornar'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:text='RETORNAR' />
    </LinearLayout>
```

8. Modificamos el archivo **listado_productos.java** para listar los productos desde la base de datos Ventas.

```
    package com.example.toshibapro.pmcaso71;

    import android.content.Intent;
    import android.support.v7.app.AppCompatActivity;
    import android.os.Bundle;
    import android.view.View; import android.widget.ArrayAdapter;
    import android.widget.Button;
    import android.widget.ListView;

    import java.util.ArrayList;

    public class listado_productos extends AppCompatActivity implements View.
    OnClickListener {

        ListView lvProductos;
        Button btRetornar;

        DAOProducto objDAO;
        ArrayList<Producto> aProductos;
        ArrayAdapter<Producto> adaptador;

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

            lvProductos = (ListView) findViewById(R.id.lvProductos);
            btRetornar = (Button) findViewById(R.id.btRetornar);
            btRetornar.setOnClickListener(this);
```

```
            listaProductos();
        }

        @Override
        public void onClick(View v) {
            Intent intent = new Intent(listado_productos.this,MainActivity.
class);
            startActivity(intent);
        }

        //Metodo que lista los productos
        public void listaProductos(){
            objDAO = new DAOProducto(this);
            aProductos = objDAO.listadoProductos();
            objDAO.close();

            adaptador = new ArrayAdapter<Producto>(this,
                                android.R.layout.simple_list_item_1,aProductos);
            lvProductos.setAdapter(adaptador);
        }
    }
```

CASO DESARROLLADO 02: CRUD para clientes

Implementemos una aplicación móvil en Android que permita gestionar el CRUD de la tabla Cliente de la base de datos Ventas. Se iniciará con una pantalla de presentación y un toolbar que muestre los íconos **Listar** y **Registrar** a partir del cual se deberá mostrar el listado de clientes y el registro de clientes, respectivamente. Para poder modificar o eliminar los datos de un determinado cliente, debemos seleccionarlo desde el listado; los datos del cliente seleccionado se mostrarán en una actividad de modificación para, finalmente, realizar las modificaciones necesarias y grabarlo. En el caso de la eliminación, sucederá algo similar: se seleccionará desde la lista y desde la misma ventana de actualización se podrá eliminar mediante un botón. Los prototipos del mantenimiento de Clientes se muestran de la siguiente manera:



Pantalla Inicial



Mantenimiento de Clientes

Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 59):

Tabla 59. Especificación de valores del proyecto

Create Android Project	Application Name : pmCaso72 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_07
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos un archivo de recurso al proyecto (clic derecho en **App > New > Android Resource File > Tipo=Menu > Nombre=menu**) y colocamos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:title="LISTADO"
        android:id="@+id/listar"
        android:icon="@drawable/listado"
        app:showAsAction="always"
        />
    <item
        android:title="REGISTRAR"
        android:id="@+id/registrar"
        android:icon="@drawable/nuevo"
        app:showAsAction="always"
        />
</menu>
```

4. Agregamos imágenes al proyecto, que servirán como íconos en el toolbar. Para ello, solo debemos copiar imágenes y pegarlas dentro de la carpeta **drawable** de la aplicación.

5. Diseñaremos el prototipo de la pantalla inicial de la aplicación en el **archivo activity_main.xml**; aquí debemos agregar el control ToolBar, desde el cual se podrá controlar el listado y el registro de clientes.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        app:title="GESTIÓN DE CLIENTES"
        app:titleTextColor="#fff"
        app:logo="@drawable/modificar" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="0dp"
        android:layout_marginTop="124dp"
        android:gravity="center"
        android:textSize="30dp"
        android:text="MANTENIMIENTO DE CLIENTES" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="229dp"
        android:layout_height="237dp"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="21dp"
        app:srcCompat="@drawable/logocentral" />

</RelativeLayout>
```

6. Modificamos el archivo **style.xml** para no mostrar la barra de título, ya que el toolbar que añadimos al proyecto será el que contenga las opciones para el CRUD del cliente. Asimismo, realizamos las siguientes modificaciones:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
```

```
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

7. Agregamos un Blank Activity llamado listado_clientes.xml, el cual definirá el listado de los clientes. Principalmente, se compondrá de un control ListView, tal como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".listado_clientes">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:layout_marginBottom="10dp"
        android:layout_marginTop="10dp"
        android:gravity="center"
        android:text="LISTADO DE CLIENTES" />

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

    <Button
        android:id="@+id/btRetornar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RETORNAR" />
</LinearLayout>
```

8. Luego, agregamos un Blank Activity **llamado registra_cliente.xml**, el cual definirá el registro de un nuevo cliente, tal como se aprecia en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".registra_cliente">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
```

```
        android:gravity="center"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:text="MANTENIMIENTO DE CLIENTES" />

    <EditText
        android:id="@+id/etCodigo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Codigo"
        android:inputType="number" />

    <EditText
        android:id="@+id/etNombres"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Nombres"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/etApellidos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Apellidos"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/etFono"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Telefono"
        android:inputType="phone" />

    <Button
        android:id="@+id/btGuardar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="GUARDAR" />

    <Button
        android:id="@+id/btRetorno"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RETORNAR" />

</LinearLayout>
```

9. Posteriormente, agregamos un Blank Activity llamado modifica_cliente.xml, el cual definirá la pantalla de actualización de un cliente seleccionado desde la lista de clientes, tal como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".modifica_cliente">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="20dp"
            android:gravity="center"
            android:layout_marginTop="10dp"
            android:layout_marginBottom="10dp"
            android:text="MANTENIMIENTO DE CLIENTES" />
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="20dp"
            android:gravity="center"
            android:layout_marginTop="10dp"
            android:layout_marginBottom="10dp"
            android:text="(ACTUALIZACIÓN DE DATOS)" />

        <EditText
            android:id="@+id/etCodigo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Codigo"
            android:inputType="number" />

        <EditText
            android:id="@+id/etNombres"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Nombres"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/etApellidos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Apellidos"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/etFono"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Telefono"
            android:inputType="phone" />

        <Button
            android:id="@+id/btActualizar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="ACTUALIZAR" />

        <Button
            android:id="@+id/btEliminar"
```

```
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:text='ELIMINAR' />

        <Button
            android:id='@+id/btRetorna'
            android:layout_width='match_parent'
            android:layout_height='wrap_content'
            android:text='RETORNAR' />
    </LinearLayout>
```

10. Agregue la clase **cliente.java**, la cual define los atributos necesarios para el mantenimiento de cliente. Debemos considerar que los atributos deben tener relación con las columnas de la tabla cliente, tal como se observa en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

public class Cliente {
    private int codigo;
    private String nombres;
    private String apellidos;
    private String telefono;

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}
```

11. Agregamos la clase **DAOCliente.java**, la cual define los métodos que permitirán realizar el mantenimiento de los clientes, así como crear la base de datos Ventas y la tabla Cliente, como se aprecia en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.widget.Toast;

import java.util.ArrayList;

public class DAOCliente extends SQLiteOpenHelper {

    //Definicion de la base de datos VENTAS y la tabla CLIENTE
    private static final String base='VENTAS';
    private static finall int version=1;
    private static final String tabla='CLIENTE';

    //Definicion de las columnas
    private static final String cod_cli='COD_CLI';
    private static final String nom_cli='NOM_CLI';
    private static final String ape_cli='APE_CLI';
    private static finalString fon_cli='FON_CLI';

    public DAOCliente(Context context) {
        super(context, base, null, version);
    }

    //Metodo para crear la tabla Cliente
    @Override
    public void onCreate(SQLiteDatabase db) {
        String SQL='CREATE TABLE '+tabla+' ('+cod_cli+' text PRIMARY KEY,'
                +nom_cli+' text,'+ape_cli+' text,'
                +fon_cli+' text)';
        db.execSQL(SQL);
    }

    //Metodo que valida la existencia de la tabla Cliente
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
        db.execSQL('drop table if exists '+tabla);
        onCreate(db);
    }

    //Metodo para agregar un cliente
    public void nuevoCliente(Cliente objC){

        //Obtener los valores a registrar
        ContentValues values=new ContentValues();
        values.put(cod_cli,objC.getCodigo());
        values.put(nom_cli,objC.getNombres());
        values.put(ape_cli,objC.getApellidos());
        values.put(fon_cli,objC.getTelefono());

        //Ejecutar el metodo Insert
        SQLiteDatabase db=getWritableDatabase();
```

```
        db.insert(tabla,null,values);

        //Cerrar la conexion
        db.close();
    }

    //Metodo para listar los clientes
    public ArrayList listaClientes(){
        ArrayList listaClientes = new ArrayList();
        SQLiteDatabase db=getWritableDatabase();

        //Aplicar la sentencia de seleccion de registros de clientes
        String SQL='SELECT * FROM '+tabla;
        Cursor c=db.rawQuery(SQL,null);

        //Concatenar la informacion obtenida de la BD
        if(c.moveToFirst()) {
            do {
                listaClientes.add(c.getInt(0)+' '+c.getString(1)+' '
                        +c.getString(2)+' '+c.getString(3));
            }while(c.moveToNext());
        }
        return listaClientes;
    }

    //Metodo para actualizar un cliente
    public void actualizaCliente(Cliente objC){

        //Obtener informacion del cliente a modificar
        int cod=objC.getCodigo();
        String nom=objC.getNombres();
        String ape=objC.getApellidos();
        String tel=objC.getTelefono();

        //Ejecutando la sentencia UPDATE
        SQLiteDatabase db=getWritableDatabase();
            db.execSQL('update cliente set nom_cli=''+nom+'',ape_
cli=''+ape+'',fon_cli=''+tel+'' where cod_cli='+cod);
        db.close();
    }

    //Metodo para eliminar un registro de cliente
    public void eliminaCliente(Cliente objC){

        //Obtener el codigo del cliente a eliminar
        int cod=objC.getCodigo();

        //Ejecutando la sentencia DELETE
        SQLiteDatabase db=getWritableDatabase();
        db.execSQL('delete from cliente where cod_cli='+cod);
        db.close();
    }

}
```

12. Modificamos la clase **MainActivity.java**, de tal forma que habilite las opciones del control ToolBar. De este modo, al seleccionarlas, nos llevarán al listado de clientes y al registro de estos, como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

import android.content.ContentValues;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import android.support.v7.widget.Toolbar;

public class MainActivity extends AppCompatActivity {

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

        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

    }

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id=item.getItemId();
        if(id==R.id.registrar)
            startActivity(new Intent(MainActivity.this,registra_cliente.
class));
        else if(id==R.id.listar) {
            startActivity(new Intent(MainActivity.this,listado_clientes.
class));
        }
        return true;
    }
}
```

13. Modificamos la clase **listado_clientes.java**, de tal forma que muestre los datos de los clientes y que, al seleccionar un cliente, nos lleve a la actividad de actualización de datos, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MenuItem;
```

```
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;

import javax.xml.transform.dom.DOMLocator;

public class listado_clientes extends AppCompatActivity {

    ListView lvClientes;
    ArrayList<Cliente> aClientes;
    ArrayAdapter<Cliente> adaptador;

    DAOCliente objDAO;

    Button btRetornar;

    @Override
    protected void onPostResume() {
        super.onPostResume();
        cargaListado();
    }

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

        btRetornar = (Button)findViewById(R.id.btRetornar);
        btRetornar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              startActivity(new Intent(listado_clientes.this,MainActivity.
class));
            }
        });

        lvClientes = (ListView)findViewById(R.id.lvClientes);

        cargaListado();

              lvClientes.setOnItemClickListener(new  AdapterView.
OnItemClickListener() {
            @Override
                public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                Toast.makeText(listado_clientes.this,objDAO.listaClientes().
get(position).toString(),Toast.LENGTH_SHORT).show();

                    int codigo=Integer.parseInt(objDAO.listaClientes().
get(position).toString().split(' ')[0]);
                   String nombres=objDAO.listaClientes().get(position).
toString().split(' ')[1];
                  String apellidos=objDAO.listaClientes().get(position).
toString().split(' ')[2];
                   String telefono=objDAO.listaClientes().get(position).
toString().split(' ')[3];
```

```
                Intent intent = new Intent(listado_clientes.this,modifica_
cliente.class);
                intent.putExtra('cod',codigo);
                intent.putExtra('nom',nombres);
                intent.putExtra('ape',apellidos);
                intent.putExtra('tel',telefono);
                startActivity(intent);
            }
        });

        if(getSupportActionBar()!=null){
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            getSupportActionBar().setDisplayShowHomeEnabled(true);
        }
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if(item.getItemId() == android.R.id.home){
            finish();
        }
        return super.onOptionsItemSelected(item);
    }

    public void cargaListado(){
        objDAO = new DAOCliente(this);
        aClientes= objDAO.listaClientes();
        objDAO.close();

        adaptador = new ArrayAdapter<Cliente>(this,
                android.R.layout.simple_list_item_1,aClientes);
        lvClientes.setAdapter(adaptador);
    }
}
```

14. Modificamos la clase **registra_cliente.java**, de tal modo que registre los datos del cliente en su respectiva tabla, tal como se observa en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class registra_cliente extends AppCompatActivity {

    Button btGuardar,btRetorno;
    EditText etCodigo,etNombres,etApellidos, etFono;
    DAOCliente objDAO=new DAOCliente(this);;

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

        etCodigo = (EditText)findViewById(R.id.etCodigo);
```

```
        etNombres = (EditText)findViewById(R.id.etNombres);
        etApellidos = (EditText)findViewById(R.id.etApellidos);
        etFono = (EditText)findViewById(R.id.etFono);

        btGuardar = (Button) findViewById(R.id.btGuardar);
        btGuardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              grabar(Integer.parseInt(etCodigo.getText().toString()),etNombres.
getText().toString(),etApellidos.getText().toString(),etFono.getText().
toString());
              startActivity(new Intent(registra_cliente.this,MainActivity.
class));
            }
        });

        btRetorno = (Button) findViewById(R.id.btRetorno);
        btRetorno.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              startActivity(new Intent(registra_cliente.this,MainActivity.
class));
            }
        });
    }

    //Metodo para grabar un cliente
    public void grabar(int codigo,String nombres, String apellidos, String
telefono){
        Cliente objC=new Cliente();
        objC.setCodigo(codigo);
        objC.setNombres(nombres);
        objC.setApellidos(apellidos);
        objC.setTelefono(telefono);
        objDAO.nuevoCliente(objC);
        Toast.makeText(this, 'Cliente registrado..!!',Toast.LENGTH_SHORT).
show();
    }
}
```

15. Modificamos la clase **modifica_cliente.java**, de tal forma que actualice y elimine los datos de un determinado cliente, tal como se muestra en el siguiente código:

```
package com.example.toshibapro.pmcaso72;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class modifica_cliente extends AppCompatActivity {

    Button btActualizar,btEliminar, btRetorna;
    EditText etCodigo,etNombres,etApellidos, etFono;
    DAOCliente objDAO;

    int cod;
```

```
    String nom,ape,tel;

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

        btRetorna = (Button)findViewById(R.id.btRetorna);
        btRetorna.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(modifica_cliente.this,MainActivity.
class));
            }
        });

        Bundle b=getIntent().getExtras();
        if (b !=null){
            cod=b.getInt('cod');
            nom=b.getString('nom');
            ape=b.getString('ape');
            tel=b.getString('tel');
        }

        etCodigo = (EditText)findViewById(R.id.etCodigo);
        etNombres = (EditText)findViewById(R.id.etNombres);
        etApellidos = (EditText)findViewById(R.id.etApellidos);
        etFono = (EditText)findViewById(R.id.etFono);

        etCodigo.setText(''+cod);
        etNombres.setText(nom);
        etApellidos.setText(ape);
        etFono.setText(tel);

        //Invocacion del metodo de actualizacion
        btActualizar = (Button) findViewById(R.id.btActualizar);
        btActualizar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                modificar(cod,etNombres.getText().toString(),etApellidos.
getText().toString(),etFono.getText().toString());
            }
        });

        //Invocacion del metodo de eliminacion
        btEliminar = (Button)findViewById(R.id.btEliminar);
        btEliminar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                eliminar(cod);
            }
        });

    }

    //Metodo para modificar un cliente
    public void modificar(int codigo,String nombres, String apellidos,
String telefono) {

        //Enviar los datos del cliente a un objeto
        Cliente objC=new Cliente();
```

```
            objC.setCodigo(codigo);
            objC.setNombres(nombres);
            objC.setApellidos(apellidos);
            objC.setTelefono(telefono);

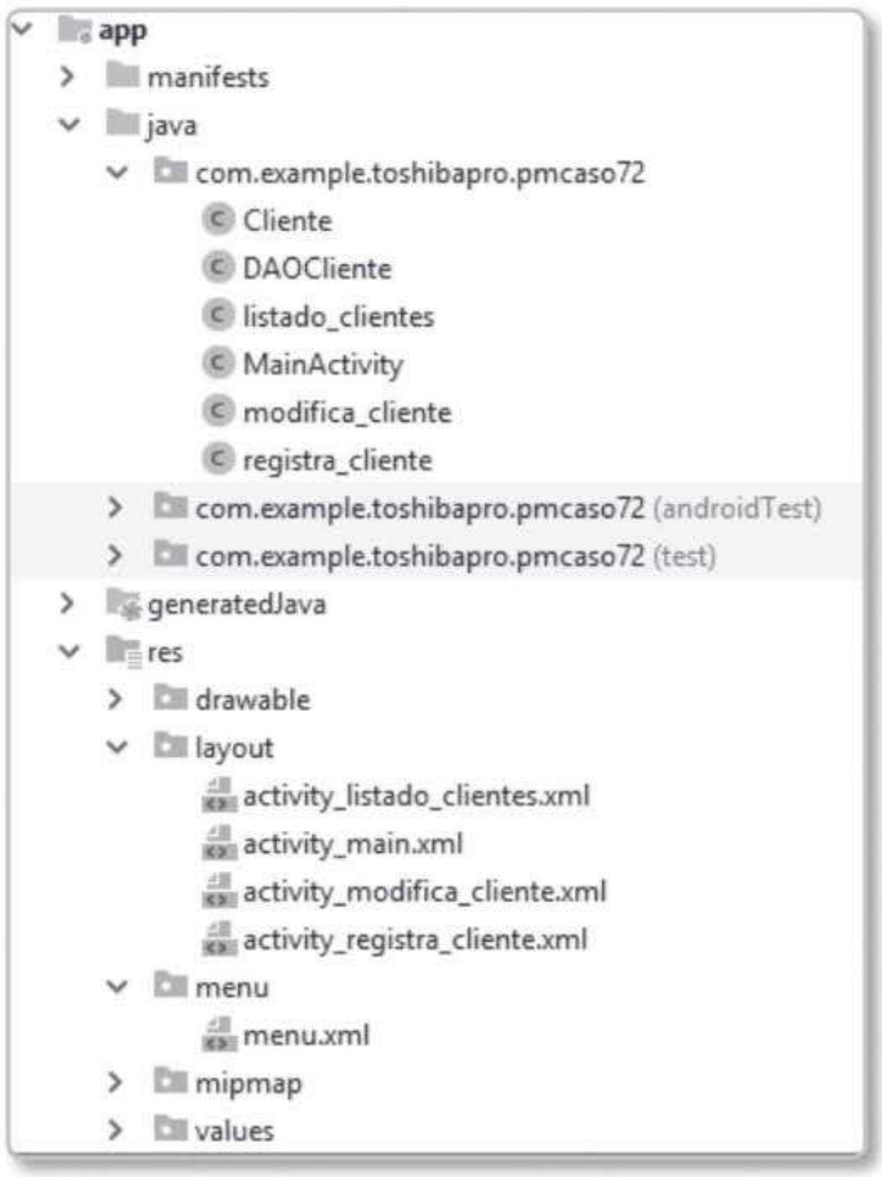
            //Invocar al metodo actualiza cliente de DAOCliente
            new DAOCliente(this).actualizaCliente(objC);
             Toast.makeText(this,'Actualizado correctamente',Toast.LENGTH_
SHORT).show();
             startActivity(new Intent(modifica_cliente.this,listado_clientes.
class));
        }

        //Metodo para eliminar un cliente
        public void eliminar(int codigo){

            //Enviar el codigo del cliente a un objeto
            Cliente objC=new Cliente();
            objC.setCodigo(codigo);

            //Invocar al metodo eliminar cliente de DAOCliente
            new DAOCliente(this).eliminaCliente(objC);
           Toast.makeText(this,'Eliminado correctamente',Toast.LENGTH_SHORT).
show();
             startActivity(new Intent(modifica_cliente.this,listado_clientes.
class));
        }
}
```

Finalmente, el explorador del proyecto debe mostrarse de la siguiente manera:



CAPÍTULO

8



Networking y servicios en Android

8.1 Qué es un servicio

Es un componente que puede realizar operaciones en primer y segundo plano. Comúnmente, no tiene interfaz; es decir, trabaja bajo una lógica. La principal característica de un servicio es que permite cambiar de una aplicación a otra y que solo finalizará cuando el servicio sea destruido. Por otro lado, podemos decir que un servicio puede realizar cualquier tipo de acción como actualizar datos, mostrar notificaciones o mostrar activities.

El sistema operativo Android incluye una serie de servicios, por ejemplo, mensajeria, notificaciones de bateria baja o actualizaciones de aplicaciones. Finalmente, para usar un servicio en una aplicación se debe referenciar en el archivo AndroidManifest.

En la Figura 33, se aprecia el ciclo de vida de un servicio.

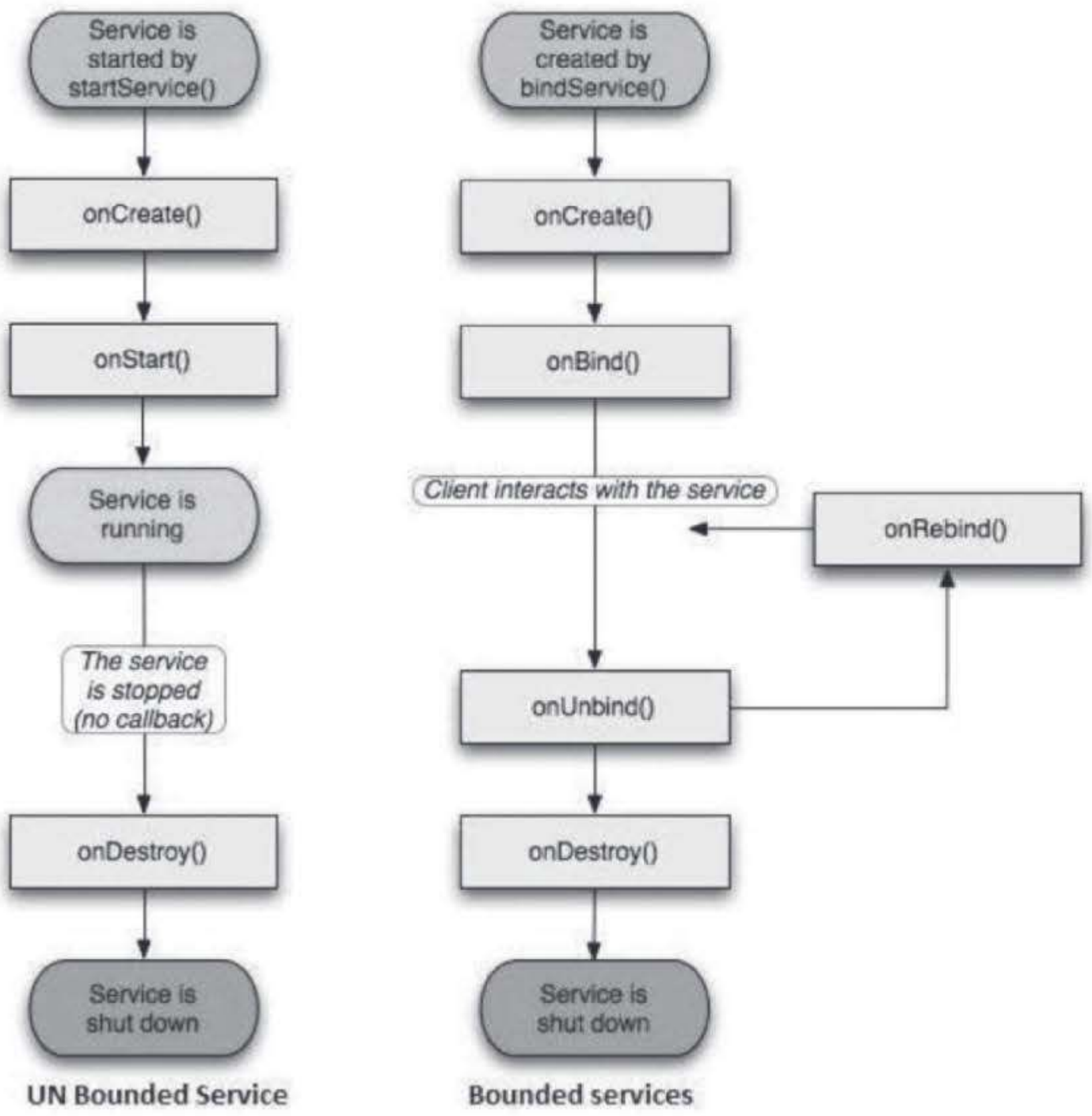


Figura 33. Ciclo de vida de una Service. Tomado de "Android-Services," por TutorialPoins, s.f. Recuperado de https://www.tutorialspoint.com/android/android_services.htm.

Por otro lado, existen dos tipos de servicios: (a) el servicio iniciado por startService, y (b) el servicio asociado con bindService o más conocido como Bound Service.

Veamos la implementación del método startService:

```
private void startService() {
  startService(<Proceso o Intencion>);
}
```

En la Tabla 60, se observa los principales métodos de la clase Service:

Tabla 60. Métodos de la clase Service

MÉTODO	DESCRIPCIÓN
onCreate()	Su ejecución evita que se vuelva a llamar otra vez al mismo servicio, pues en esta etapa el servicio ya debe estar iniciado en la memoria del dispositivo.
onStartCommand()	Tiene la finalidad de ejecutar sentencias del servicio única y exclusivamente si el servicio fue iniciado con startService().
onBind()	Solo se ejecuta si el servicio está asociado con bindService() por un componente. Asimismo, devuelve una interfaz Java de comunicación del tipo IBinder. Este método siempre debe invocarse, incluso dentro de los started services, los cuales siempre retornan null.
onDestroy()	Tiene la finalidad se destruir el servicio. Si hay asociados hilos (thread), primero deberán detenerse.

Veamos la implementación del método onCreate:

```
public void onCreate(){
  super.onCreate();
}
```

Veamos la implementación del método onStartCommand:

```
@Override
public int onStartCommand(Intent intent,int flags,int startId){
   return super.onStartCommand(intent,flags,startId);
}
```

Veamos la implementación del método onStartCommand:

```
@Override
public void onDestroy() {
  super.onDestroy();
}
```

8.2 Intent Service

Es una subclase de la clase Service que permite ejecutar un proceso de un conjunto de solicitudes uno por vez. Esto evita la concurrencia de procesos desde un mismo servicio. El método onHandleIntent es el que recibe la solicitud de inicio para trabajar el servicio en segundo plano. Los pasos para la ejecución de un servicio se pueden visualizar en el siguiente ciclo (ver Figura 34):

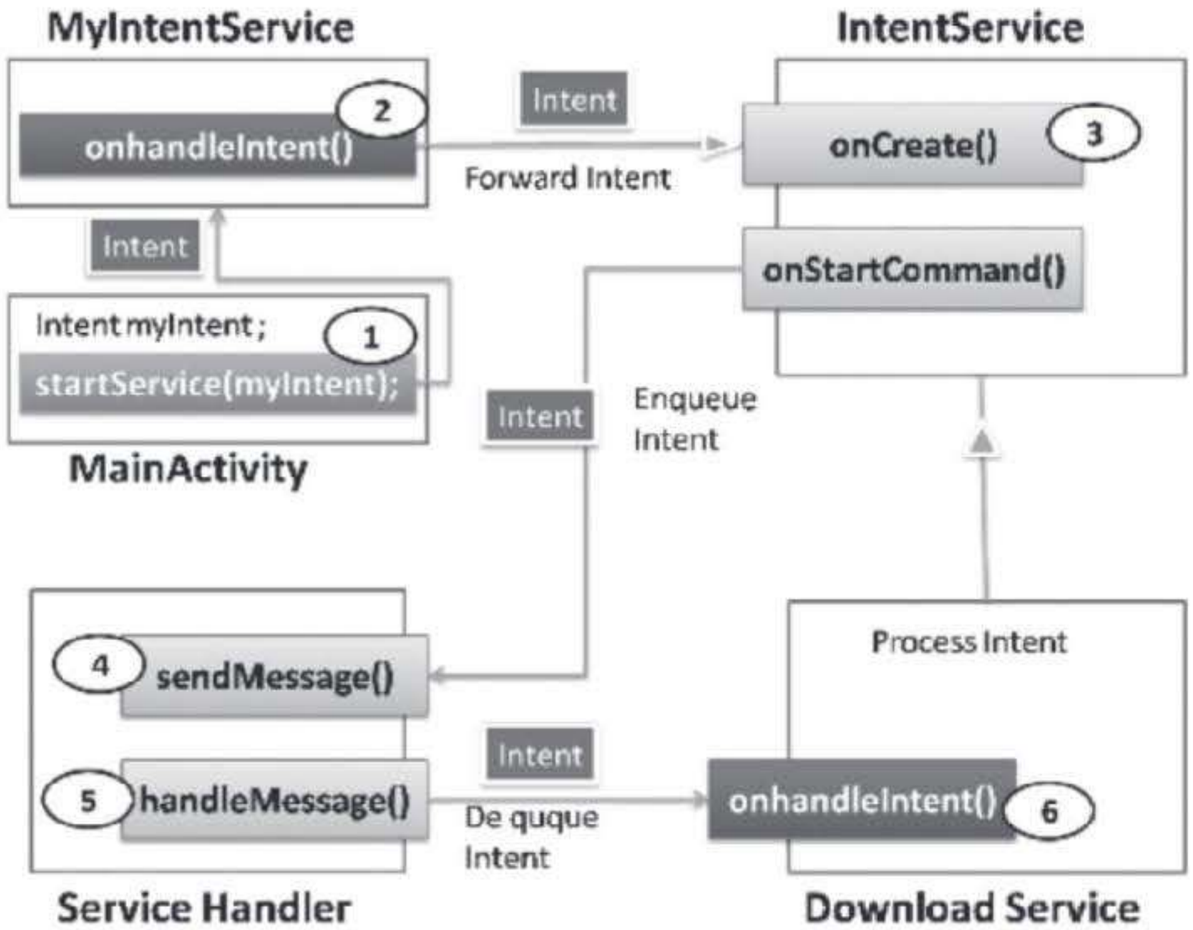


Figura 34. Ciclo de ejecución de un Service. Tomado de "Android Intent Service," por Android Concepts, 2014. Recuperado de http://androidconceptspp.blogspot.com/2014/06/android-intent-service.html.

8.3 Bound Service

Es usado para servicios de enlace y se ejecuta solo mientras otro componente está enlazado a él. Su uso se basa en el metodo BindService, el cual presenta una Interfaz cliente-servidor que permite que los componentes interactúen con el servicio recibiendo y enviando mensajes.

Veamos el siguiente código, que usa métodos que permiten usar la clase Bound del servicio:

```
boolean isBound-false;
private ServiceConnection serviceConnection-new ServiceConnection() {
@Override
  public void onServiceConnected(ComponentName name, IBinder service) {
   MyService.LocalBinder binder-(MyService.LocalBinder)service;
           miServicio- binder.getService();
           isBound - true;
  }

  @Override
  public void onServiceDisconnected(ComponentName name) {
           isBound - false;
  }
};
```

8.4 Configurar el archivo de manifiesto

Al igual que las actividades (y otros componentes), debemos declarar todos los servicios en el archivo de manifiesto de la aplicación.

Para declarar el servicio, agregamos un elemento <**service**> como campo secundario del elemento de <**application**> . Por ejemplo:

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android='http://schemas.android.com/apk/res/android'
    package='com.example.toshibapro.pmcaso81'>

    <application
        <activity android:name='.MainActivity'>
            <intent-filter>
                <action android:name='android.intent.action.MAIN' />

                <category android:name='android.intent.category.LAUNCHER' />
            </intent-filter>
        </activity>
        <service android:name='.nombre_del_servicio'></service>
    </application>
</manifest>
```

CASO DESARROLLADO 01: servicios de mensajes

Implementemos una aplicación móvil que permita iniciar un servicio y que, bajo el tiempo de 20 segundos, solicite el ingreso de un mensaje que será enviado al servicio y que este devolverá en la cadena invertida. Pasados los 20 segundos, automáticamente se destruye el servicio y, a partir de ese momento, ya no podrá enviar el mensaje. No olvidemos que se debe iniciar el servicio mediante un botón.



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 61):

Tabla 61. Especificación de valores del proyecto

Create Android Project	Application Name Company Domain Project Location	: pmCaso81 : Modifique por un nombre adecuado : C:\Libro Android\Capitulo_08
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar Activity Name Layout Name	: Empty Activity : MainActivity : activity_main

3. Modificamos el archivo **activity_main.xml** para diseñar el protipo de la aplicación.

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <Button
        android:id='@+id/btIniciar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='INICIAR SERVICIO' />

    <EditText
        android:id='@+id/etMensaje'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:ems='10'
        android:hint='Mensaje a enviar'
        android:inputType='textPersonName' />

    <Button
        android:id='@+id/btEnviar'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:text='ENVIAR MENSAJE A SERVICIO' />

    <EditText
        android:id='@+id/cntreceiver'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:ems='10'
        android:inputType='textPersonName' />

    <EditText
        android:id='@+id/msgreceiver'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:ems='10'
        android:inputType='textPersonName' />
</LinearLayout>
```

4. Agregamos una clase al proyecto llamado **Intent_Service** y colocamos el siguiente código perteneciente a los procesos del servicio:

```
package com.example.toshibapro.pmcaso81;

import android.app.IntentService;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Looper;
import android.support.annotation.Nullable;
```

```
import android.widget.Toast;

public class Intent_Service extends IntentService {

    //Definicion de valores constantes
    final static String KEY_INT_FROM_SERVICE='KEY_INT_FROM_SERVICE';
    final static String KEY_STRING_FROM_SERVICE='KEY_STRING_FROM_SERVICE';
    final static String ACTION_UPDATE_CNT='UPDATE_CNT';
    final static String ACTION_UPDATE_MSG='UPDATE_MSG';

    final static String KEY_MSG_TO_SERVICE='KEY_MSG_TO_SERVICE';
    final static String ACTION_MSG_TO_SERVICE='MSG_TO_SERVICE';

    MiReceptorPrincipal miReceptorPrincipal;
    int cnt;

    //Metodo Constructor
    public Intent_Service() {
        super('Intent_Service');
    }


    @Override
    protected void onHandleIntent(Intent intent) {
        String prompt;
        //Evaluando el hilo
        if(Looper.myLooper() == Looper.getMainLooper()){
            prompt = 'Tiempo Fuera...!!!';
        }else{
            prompt = 'Esperando ingreso de mensaje..!!';
        }

        Intent iPrompt = new Intent();
        iPrompt.setAction(ACTION_UPDATE_MSG);
        iPrompt.putExtra(KEY_STRING_FROM_SERVICE, prompt);
        sendBroadcast(iPrompt);

        //Definir 20 segundos
        cnt = 20;
        while (cnt >= 0){
            try {
                Thread.sleep(1000);

                Intent i = new Intent();
                i.setAction(ACTION_UPDATE_CNT);
                i.putExtra(KEY_INT_FROM_SERVICE, cnt);
                sendBroadcast(i);

                cnt--;
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void onCreate(){
        Toast.makeText(getApplicationContext(),'onCreate',Toast.LENGTH_
SHORT).show();
        miReceptorPrincipal = new MiReceptorPrincipal();
        super.onCreate();
    }
```

```
    @Override
    public int onStartCommand(Intent intent,int flags,int startId){
     Toast.makeText(getApplicationContext(),'onStartCommand',Toast.LENGTH_
SHORT).show();
        IntentFilter intentFilter=new IntentFilter();
        intentFilter.addAction(ACTION_MSG_TO_SERVICE);
        registerReceiver(miReceptorPrincipal,intentFilter);

        return super.onStartCommand(intent,flags,startId);
    }

    @Override
    public void onDestroy() {
        Toast.makeText(getApplicationContext(),'onDestroy',Toast.LENGTH_
SHORT).show();
        unregisterReceiver(miReceptorPrincipal);
        super.onDestroy();
    }

    //Definicion de MiReceptorPrincipal
    public class MiReceptorPrincipal extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if(action.equals(ACTION_MSG_TO_SERVICE)){
                String msg=intent.getStringExtra(KEY_MSG_TO_SERVICE);
                msg=new StringBuilder(msg).reverse().toString();

                //Volviendo al MainActivity
                Intent i = new Intent();
                i.setAction(ACTION_UPDATE_MSG);
                i.putExtra(KEY_STRING_FROM_SERVICE,msg);
                sendBroadcast(i);
            }
        }
    }
}
```

5. Modificamos la clase **MainActivity.java**, de tal forma que consuma el servicio implementado. Para ello, colocamos el siguiente código:

```
package com.example.toshibapro.pmcaso81;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    Button btEnviar, btIniciar;
    EditText etMensaje, etContador, etMensajeServicio;

    MiReceptorPrincipal miReceptorPrincipal;
```

```
    Intent myIntent = null;

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

        btEnviar = (Button) findViewById(R.id.btEnviar);
        btIniciar = (Button) findViewById(R.id.btIniciar);

        etContador = (EditText) findViewById(R.id.etContador);
        etMensajeServicio = (EditText) findViewById(R.id.etMensajeServicio);
        etMensaje = (EditText) findViewById(R.id.etMensaje);

        btEnviar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String msgToService = etMensaje.getText().toString();
                Intent intent = new Intent();
                intent.setAction(Intent_Service.ACTION_MSG_TO_SERVICE);
                    intent.putExtra(Intent_Service.KEY_MSG_TO_SERVICE,
msgToService);
                sendBroadcast(intent);
            }
        });

        //Iniciar el servicio
        btIniciar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startService();
            }
        });
    }

    //Ejecutar la clase Intent_Service
    private void startService() {
        myIntent = new Intent(MainActivity.this, Intent_Service.class);
        startService(myIntent);
    }

    //Ejecutar la sentencia onStart
    @Override
    protected void onStart() {
        miReceptorPrincipal = new MiReceptorPrincipal();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(Intent_Service.ACTION_UPDATE_CNT);
        intentFilter.addAction(Intent_Service.ACTION_UPDATE_MSG);
        registerReceiver(miReceptorPrincipal, intentFilter);
        super.onStart();
    }

    //Ejecutar la sentencia onStop
    @Override
    protected void onStop() {
        unregisterReceiver(miReceptorPrincipal);
        super.onStop();
    }

    //Clase broadcastReceiver
    private class MiReceptorPrincipal extends BroadcastReceiver {
```

```
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (action.equals(Intent_Service.ACTION_UPDATE_CNT)) {
              int int_front_service = intent.getIntExtra(Intent_Service.
KEY_INT_FROM_SERVICE, 0);
                etContador.setText(String.valueOf(int_front_service));
            } else if (action.equals(Intent_Service.ACTION_UPDATE_MSG)) {
                String string_from_service =
                                    intent.getStringExtra(Intent_Service.
KEY_STRING_FROM_SERVICE);
                etMensajeServicio.setText(String.valueOf(string_from_
service));
            }
        }

    }
}
```

6. Finalmente, invocamos al servicio en el archivo **AndroidMainfest.xml** de la siguiente manera:

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android='http://schemas.android.com/apk/res/android'
    package='com.example.toshibapro.pmcaso81'>

    <application
        <activity android:name='.MainActivity'>
            <intent-filter>
                <action android:name='android.intent.action.MAIN' />

                <category android:name='android.intent.category.LAUNCHER' />
            </intent-filter>
        </activity>
        <service android:name='.nombre_del_servicio'></service>
    </application>
</manifest>
```

CASO DESARROLLADO 02: generación de números aleatorios con servicio usando Bound Service

Implementemos una aplicación móvil que permita generar números aleatorios a partir de un servicio. Dicho número se mostrará al presionar el botón **Generar Numero**. Usemos la clase Bound Service.



Solución:

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 62):

Tabla 62. Especificación de valores del proyecto

Create Android Project	Application Name : pmCaso82 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_08
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos un servicio al proyecto (clic derecho en **App > New > Service > Service**) y asignamos el nombre MyService, como se muestra en la siguiente captura:



4. Modificamos el archivo **activity_main.xml** para diseñar el prototipo de la aplicación.

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:app='http://schemas.android.com/apk/res-auto'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <Button
        android:id='@+id/btGenera'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:onClick='getGenerarNumero'
        android:text="GENERAR NUMERO' />

    <TextView
        android:id='@+id/tvNumero'
        android:layout_width='match_parent'
        android:layout_height='wrap_content'
        android:layout_marginTop='20dp'
        android:layout_marginBottom='20dp'
        android:textSize='50dp'
        android:gravity='center'
        android:text='000' />
</LinearLayout>
```

5. Agregamos una clase llamada **MyService.java** y colocamos el siguiente código, que permitirá generar números aleatorios:

```
package com.example.toshibapro.pmcaso82;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;

import java.util.Random;

public class MyService extends Service {

    //Declaracion de variables globales
    private final IBinder iBinder=new LocalBinder();
    private final Random mGenerator=new Random();

    public class LocalBinder extends Binder{
        MyService getService(){
            return MyService.this;
        }
    }

    public MyService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return iBinder;
    }

    public int getRandom(){
        return mGenerator.nextInt(200);
    }
}
```

6. Modificamos el archivo **MainActivity.java** para invocar al servicio de generación de números aleatorios.

```
package com.example.toshibapro.pmcaso82;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    Button btGenerar;
    TextView tvNumero;

    MyService miServicio;
    boolean isBound=false;

    @Override
```

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

        Intent intent=new Intent(this, MyService.class);
        bindService(intent,serviceConnection,Context.BIND_AUTO_CREATE);
    }

    public void getGenerarNumero(View v){
        tvNumero=(TextView)findViewById(R.id.tvNumero);
        tvNumero.setText(Integer.toString(miServicio.getRandom()));
    }

    //Implementacion del servicio de conexion
    private ServiceConnection serviceConnection=new ServiceConnection() {
        @Override
       public void onServiceConnected(ComponentName name, IBinder service)
{
            MyService.LocalBinder binder=(MyService.LocalBinder)service;
            miServicio= binder.getService();
            isBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };
}
```

CAPÍTULO

9



Servicios basados en localización

9.1 Google Maps

Es una aplicación desarrollada por Google, la cual se considera un servicio de información sobre la ubicación actual, así como sobre direcciones específicas, trazado de recorridos, etc. Su tecnología se basa en el servicio de posición global con que cuentan la mayoría de dispositivos móviles (ver Figura 35).

Inicialmente, se presentaba como una aplicación web y software de escritorio; luego, pasó a los dispositivos móviles para obtener información de las ubicaciones en tiempo real.

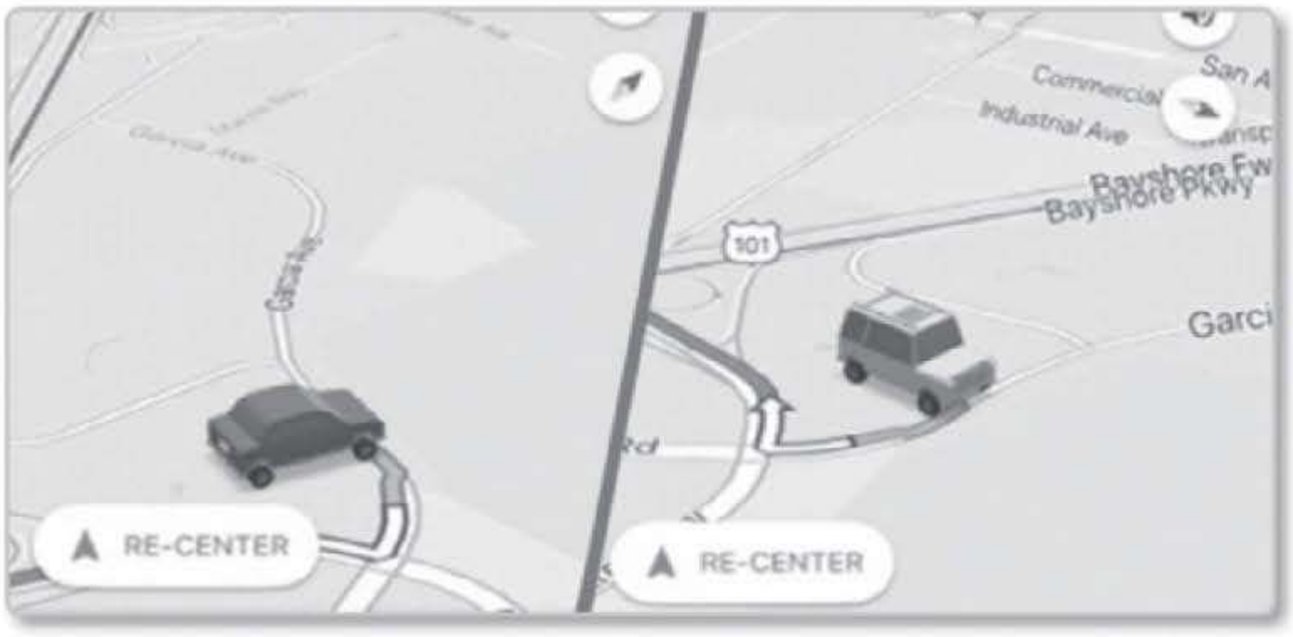


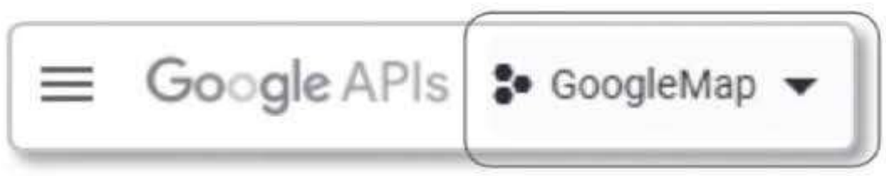
Figura 35. Funcionalidad de Google Maps. Tomado de "Google Maps cambia su indicador por carritos para guiarte," por J. Palacios, 2018. Recuperado de https://www.sopitas.com/entretenimiento/google-maps-autos-colores-actualizacion/

Entre las funciones más importantes con que cuenta Google Maps se encuentran las siguientes:

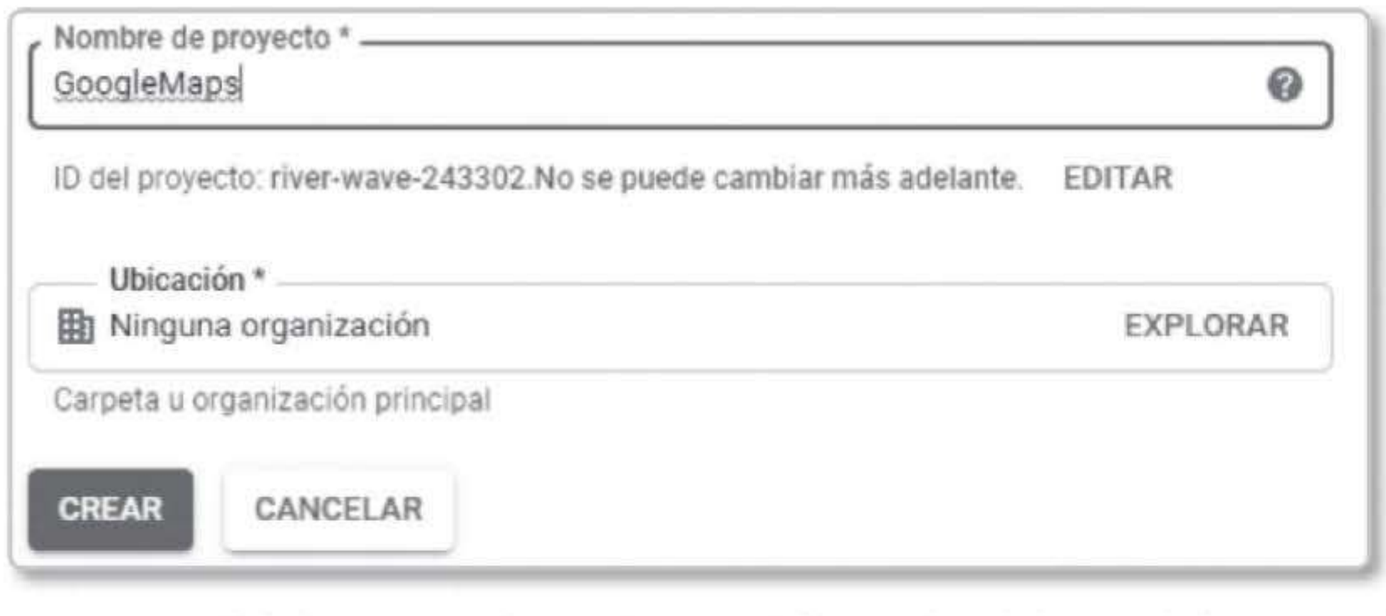
- Búsqueda de direcciones: Mediante esta función, solo debemos ingresar una dirección y la aplicación la mostrará en el plano.
- Lugares cercanos: Muestra una lista de establecimientos de servicios más cercanos como grifos, tiendas, escuelas, etc.
- Próximos eventos: Muestra una lista de eventos que están programados en ciertos establecimientos, comúnmente para eventos públicos.

9.2 Manejo de las API de Google Maps

Vamos a iniciar creando una clave API, la cual nos permitirá asociarnos al Maps SDK para Android. Para esto, ingresamos al URL: **https://console.developers.google.com**. Previamente, debemos loguearnos con una cuenta propia en Gmail y crear un nuevo proyecto, tal como se muestra en la siguiente captura:



Luego, seleccionamos **Nuevo Proyecto**, agregamos un nombre al proyecto y presionamos el botón **CREAR**, tal como se muestra en la siguiente imagen:



Luego, seleccionamos **Biblioteca** y añadimos **Maps SDK for Android** dentro del proyecto GoogleMaps.



Luego, se mostrará la ventana de biblioteca **Maps SDK for Android**, en la cual debemos presionar el botón **Habilitar**.



Seguidamente, debemos crear una credencial con la opción **Clave de API**. Seguimos los pasos que solicita y eso generará un código.



Dicho código será copiado en la aplicación. En la ventana siguiente, solo debemos presionar **Cerrar**.



Finalmente, en el proyecto, debemos agregar la siguiente dependencia:



CASO DESARROLLADO 01: **ubicación actual en el Google Maps**

Implementemos una aplicación móvil en Android que permita mostrar nuestra ubicación actual desde el Google Maps. Debemos tener en cuenta que la aplicación solo se ejecutará en un dispositivo móvil; por lo tanto, debemos crear su respectivo APK.

El prototipo de la aplicación se muestra de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 63):

Tabla 63. Especificación de valores del proyecto

Create Android Project	Application Name : pmCaso91 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_09
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Agregamos la dependencia **com.google.android.gms:play-services-maps:15.0.1** al proyecto. Para ello, debemos presionar la tecla **F4 > ficha dependencia > Clic en + > Library Dependency.**



4. Nos ubicamos en el archivo **build.dragle(app)** y colocamos el siguiente código:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.google.android.gms:play-services:10.2.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
}
```

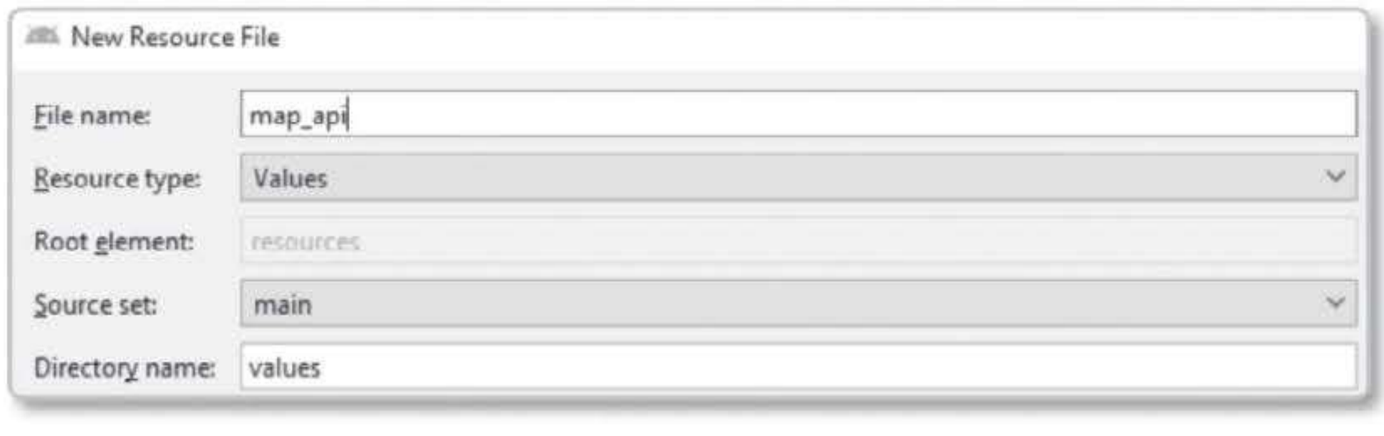
5. Preparamos el prototipo de la aplicación con el siguiente código en el archivo **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

<fragment
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    />

</android.support.constraint.ConstraintLayout>
```

6. Agregamos un **New Resource File** al proyecto (clic derecho en **app > New > Android Resource File**) y colocamos la siguiente configuración:



En el archivo **map_api.xml**, colocamos el siguiente código. No obstante, aquí debemos agregar la clave API generada en los pasos anteriores.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="map_key" translatable="false">
        AIzaSyChrh0CQ4qQ9_krU6Lw3TwoMeWIWgnEPyI
    </string>
</resources>
```

7. Modificamos el archivo **MainActivity.java** y colocamos el siguiente código:

```
package com.example.toshibapro.pmcaso91;

import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MainActivity extends FragmentActivity implements
OnMapReadyCallback {

    //Crear un objeto de la clase GoogleMap
    GoogleMap map;

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

            SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

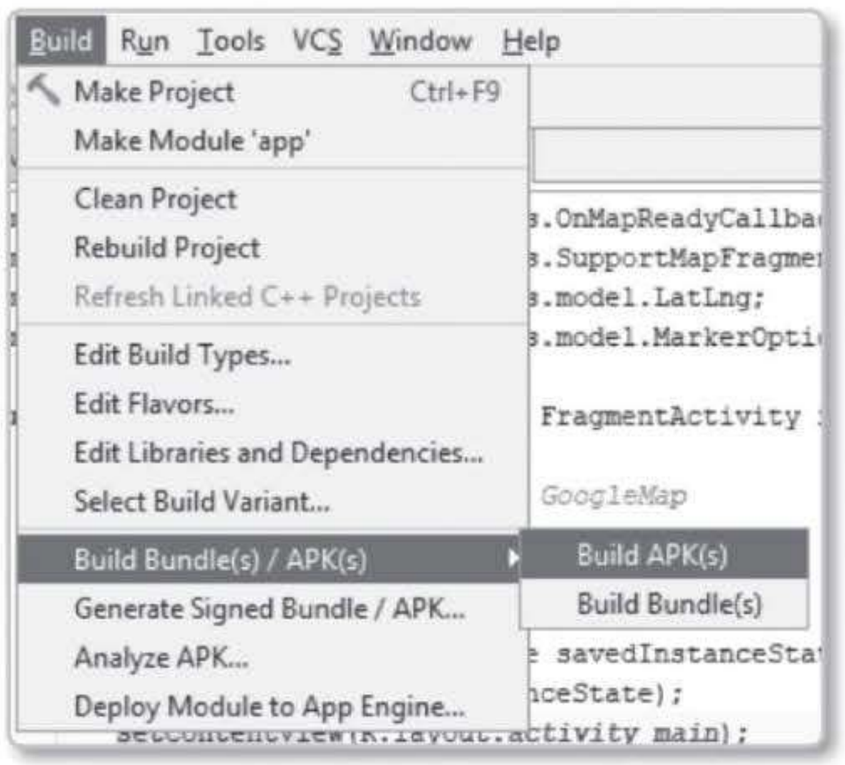
    @Override
    public void onMapReady(GoogleMap googleMap) {

        map = googleMap;
```

```
            //Colocar la ubicacion desde GoogleMaps
            LatLng miUbicacion = new LatLng(-11.916166, -77.056094);

            //Agregar un mensaje y una marca de ubicacion
            map.addMarker(new MarkerOptions().position(miUbicacion).title('Mi
Ubicacion'));
            map.moveCamera(CameraUpdateFactory.newLatLng(miUbicacion));
        }
    }
```

8. Una vez finalizada la aplicación, creamos el APK (**Build > Builds Bundle(s)> Build APK(s)**), tal como se muestra en la siguiente imagen:



9. Una vez generado el APK, lo copiamos a nuestro dispositivo móvil y lo ejecutamos para probar la aplicación.

CAPÍTULO

10



Multimedia en Android

10.1 Componente MediaPlayer

Es una clase especializada en la gestión de archivos de tipo multimedia. El objeto de la clase MediaPlayer pasa por diferentes estados (ver Tabla 64):

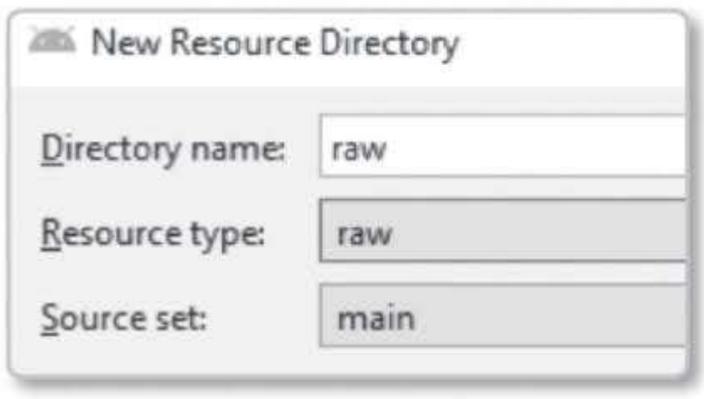
Tabla 64. Estados de la clase MediaPlayer

ESTADO	DESCRIPCIÓN
Inicializado	Initialized, el cual permite inicializar los recursos multimedia.
Preparando	Preparing, el cual permite preparar para la reproducción de audio o video.
Reproduciendo	Started reproduce el elemento multimedia.
Pausa	Pause aplica una pausa al elemento reproducido.
Parado	Stop termina el proceso de reproducción del elemento multimedia.
Reproducción completa	PlayBack Completed indica la finalización de la reproducción del elemento multimedia.
Finalizado	End libera el recurso usado para el elemento multimedia.
Error	Error aplica un corte en el proceso de reproducción.

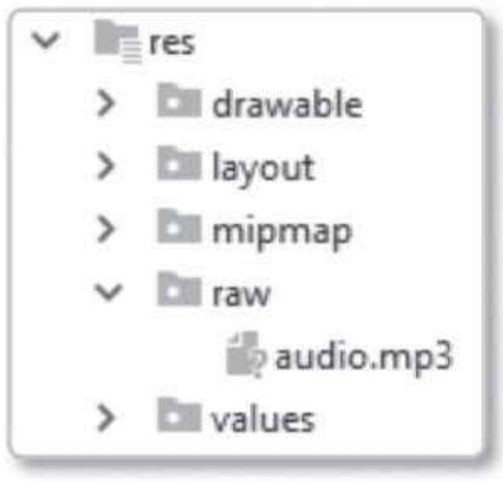
Solo debemos tener en cuenta cuándo usar los métodos, pues, para iniciar la reproducción, primero, debe encontrarse preparada, o para pausar, primero, debe reproducirse el elemento multimedia. En el caso de que aplicáramos de forma errada los métodos, se producirá un error en la aplicación y esta se cerrará.

Generalmente, para reproducir un audio o video mediante un objeto de la clase MediaPlayer, podemos seguir los siguientes pasos:

1. En el proyecto, agregamos una carpeta de recursos llamada **raw** (clic derecho en **App > New > Android Resouce Directory > Resource type: raw**), tal como se muestra en la siguiente imagen:



2. Luego, copiamos los archivos multimedia a la carpeta **raw**, ya sea con extensión .mp3 o .avi.



3. Declaramos el objeto de la clase MediaPlayer dentro del archivo .java de la aplicación:

Primera forma:

```
MediaPlayer mpAudio, mpVideo;
mpAudio = MediaPlayer.create(<ubicación_actual>,R.raw.audio);
```

Segunda forma:

```
MediaPlayer mpAudio = MediaPlayer.create(<ubicación_actual>, R.raw.audio);
```

4. Finalmente, iniciamos la reproducción con la sentencia

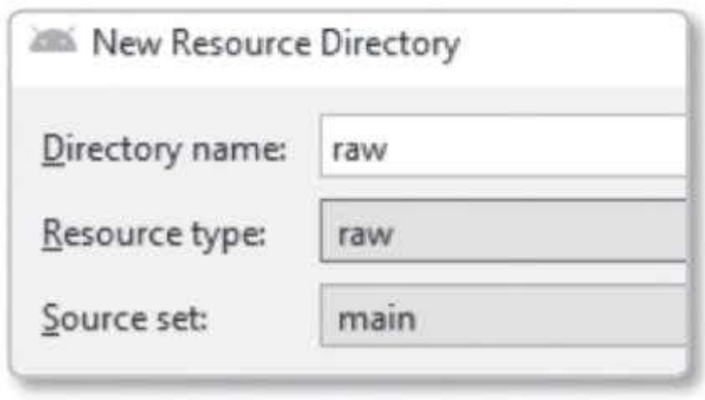
```
mpAudio.start();
```

10.2 Visualización de videos

Contamos con varias formas de visualizar videos:

Primera. Si tenemos el video dentro de nuestro dispositivo:

1. En el proyecto, agregamos una carpeta de recursos llamada **raw** (clic derecho en **App > New > Android Resouce Directory > Resource type: raw**), tal como se muestra en la siguiente captura:



2. Creamos un objeto de la clase MediaPlayer, por ejemplo:

```
MediaPlayer mPlayer;
```

3. Hacemos referencia al video con el siguiente código:

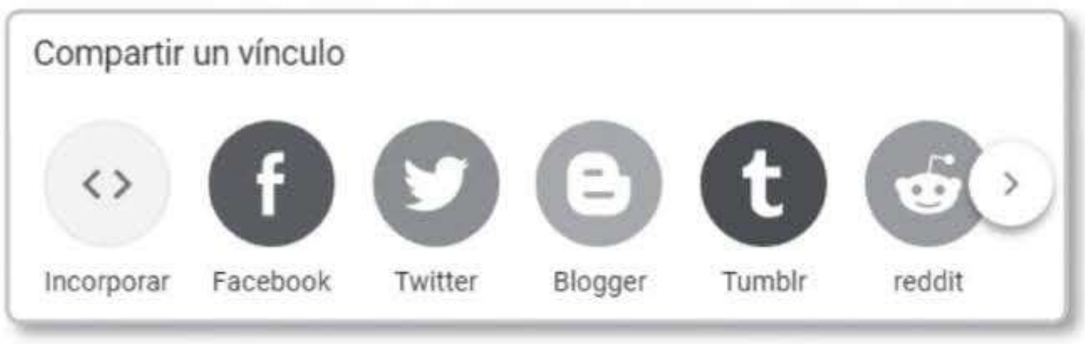
```
mPlayer=MediaPlayer.create(this, R.raw.mivideo);
mPlayer.start();
```

Segundo. Si tenemos una dirección de Youtube:

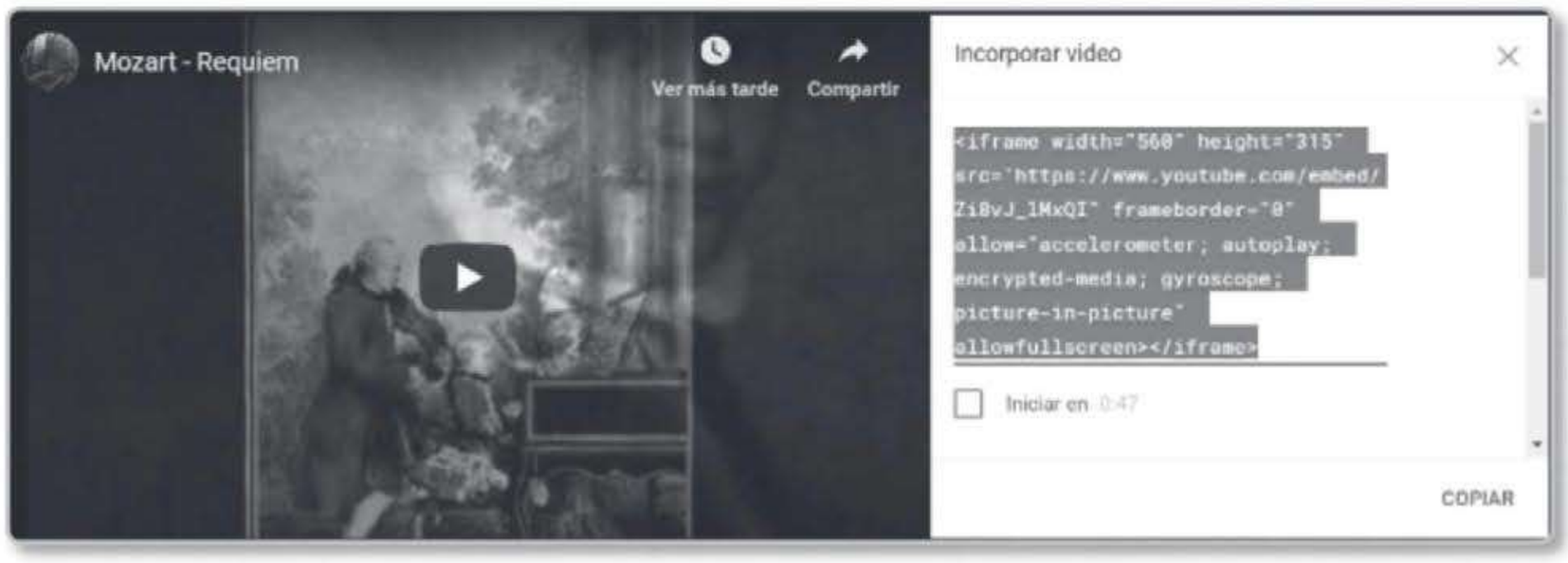
1. Obtenemos la dirección de incorporación de un determinado video de Youtube. Para esto, debemos iniciar Youtube y buscar un video. Debajo de este, se presenta el botón:



Luego, seleccionamos la opción **Incorporar**.



Luego, se mostrará la siguiente pantalla:



Finalmente, podemos seleccionar la opción **Copiar**, o seleccionar el texto y presionar las teclas "CTRL+C".

2. Creamos un objeto de la clase YouTubePlayerView:

   YouTubePlayerView youtube;

3. Finalmente, enviamos el código del video para su reproducción. Esta puede ocurrir de dos formas: la primera, por el código del video:

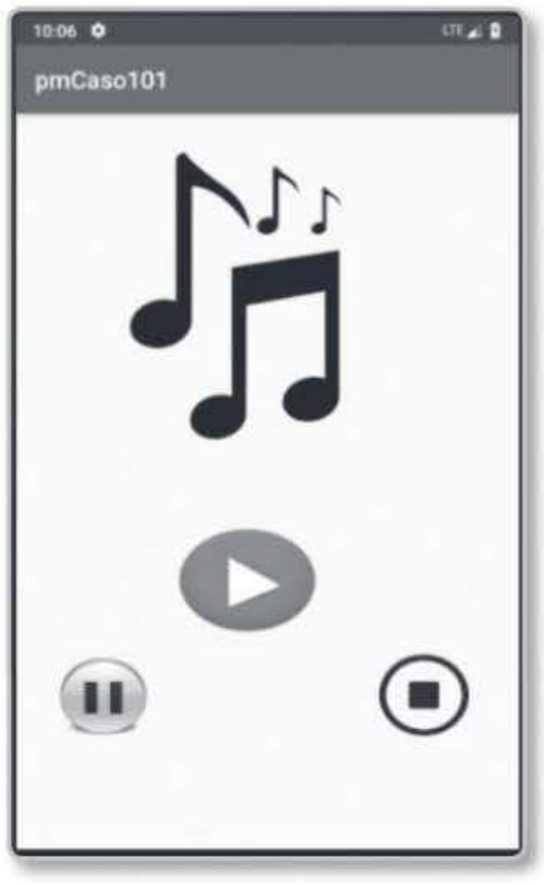
   youTubePlayer.loadVideo("yAzx8vF4Bl8");

   o también, por la dirección del video:

```
youtubeVideos.add( new YouTubeVideos("<iframe width=\"100%\" height=\"100%\" src=\"https://www.
youtube.com/embed/SY3Kxf7ZTeI\" frameborder=\"0\" allow=\"accelerometer; autoplay; encrypted-media;
gyroscope; picture-in-picture\" allowfullscreen></iframe>") );
```

CASO DESARROLLADO 01: **reproductor de audio (MP3)**

Implementemos una aplicación móvil que permita reproducir audios de tipo MP3, los cuales se encuentran dentro del dispositivo móvil. Consideremos que, si el audio inicia, se pueda pausar y volver a iniciar y continuar. En caso de aplicar para el audio, termine y vuelva a iniciar. El prototipo de la aplicación se muestra de la siguiente manera:



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 65):

Tabla 65. Especificación de valores del proyecto

Create Android Project	Application Name	: pmCaso101
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_10
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar	: Empty Activity
	Activity Name	: MainActivity
	Layout Name	: activity_main

3. Preparamos el prototipo de la aplicación con el siguiente código en el archivo **activity_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="258dp"
        android:layout_height="235dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="51dp"
        android:layout_marginTop="32dp"
        app:srcCompat="@drawable/music" />

    <Button
        android:id="@+id/btPlay"
        android:layout_width="129dp"
        android:layout_height="97dp"
        android:layout_below="@+id/imageView"
        android:layout_alignParentStart="true"
        android:layout_marginStart="127dp"
        android:layout_marginTop="68dp"
        android:background="@drawable/play" />

    <Button
        android:id="@+id/btPause"
        android:layout_width="82dp"
        android:layout_height="69dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="31dp"
        android:layout_marginBottom="91dp"
        android:background="@drawable/pause" />
```

```
        <Button
            android:id="@+id/btStop"
            android:layout_width="74dp"
            android:layout_height="67dp"
            android:layout_alignTop="@+id/btPause"
            android:layout_alignParentEnd="true"
            android:layout_marginTop="0dp"
            android:layout_marginEnd="40dp"
            android:background="@drawable/stop" />

    </RelativeLayout>
```

4. Modificamos el archivo **MainActivity.java** con el siguiente código:

```
package com.example.toshibapro.pmcaso101;

import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button btPause,btPlay,btStop;

    //Objeto de la clase MediaPlayer que iniciara el audio
    MediaPlayer mpAudio;

    //En caso se solicite pausar el audio esta variable captura la posicion
    //para volverlo a iniciar
    int posicionPausa;

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

        //Codigo de pausa sobre el audio
        btPause=(Button) findViewById(R.id.btPause);
        btPause.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mpAudio != null){
                    mpAudio.pause();
                    posicionPausa = mpAudio.getCurrentPosition();
                }
            }
        });

        //Codigo de Inicio del audio
        btPlay=(Button) findViewById(R.id.btPlay);
        btPlay.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
               if (mpAudio == null){
               mpAudio = MediaPlayer.create(getApplicationContext(),R.raw.audio);
               mpAudio.start();
               }else if(!mpAudio.isPlaying()){
```

```
                    mpAudio.seekTo(posicionPausa);
                    mpAudio.start();
                }

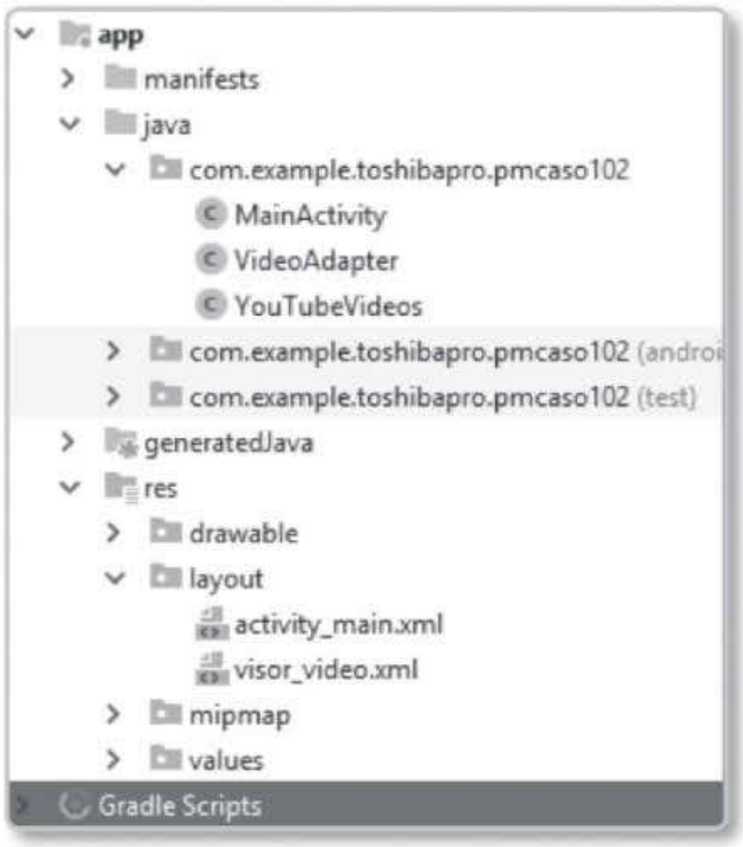
            }
        });

        //Codigo de parar el audio
        btStop=(Button) findViewById(R.id.btStop);
        btStop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mpAudio != null){
                    mpAudio.stop();
                    mpAudio=null;
                }
            }
        });

    }
}
```

CASO DESARROLLADO 02: visor de videos desde Youtube

Implementemos una aplicación móvil que permita reproducir videos desde Youtube. Los videos se obtendrán a partir de la dirección compartida de YouTube.



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 66):

Tabla 66. Especificación de valores del proyecto

Create Android Project	Application Name	: pmCaso102
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_10

Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Modificamos el archivo **activity_main.xml** para colocar el listado de videos con el control RecyclerView, tal como se muestra en el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android='http://schemas.android.com/apk/res/android'
        xmlns:app='http://schemas.android.com/apk/res-auto'
xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height='match_parent'
    android:orientation='vertical'
    tools:context='.MainActivity'>

    <android.support.v7.widget.RecyclerView
        android:id='@+id/rvVideos'
        android:layout_width="match_parent"
        android:layout_height='match_parent' />
</LinearLayout>
```

4. Agregamos un Layout al proyecto llamado visor_video.xml para definir el tamaño de la presentación de los videos. Para ello, colocamos el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<android.support.constraint.ConstraintLayout
    xmlns:android='http://schemas.android.com/apk/res/android'
    xmlns:tools='http://schemas.android.com/tools'
    android:layout_width='match_parent'
    android:layout_height="210dp'>

    <WebView
        android:id='@+id/wvVideo'
        android:layout_width='match_parent'
        android:layout_height='210dp' />
</android.support.constraint.ConstraintLayout>
```

5. Agregamos la clase YouTubeVideos.java y colocamos el siguiente código :

```
package com.example.toshibapro.pmcaso102;

public class YouTubeVideos {
    String videoUrl;

    public YouTubeVideos() {
    }

    public YouTubeVideos(String videoUrl) {
        this.videoUrl = videoUrl;
    }

    public String getVideoUrl() {
```

```
        return videoUrl;
    }

    public void setVideoUrl(String videoUrl) {
        this.videoUrl = videoUrl;
    }
}
```

6. Agregamos la clase **VideoAdapter.java** y colocamos el siguiente código:

```
package com.example.toshibapro.pmcaso102;

import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.webkit.WebChromeClient;
import android.webkit.WebView;

import java.util.List;

public class VideoAdapter extends RecyclerView.Adapter<VideoAdapter.
VideoViewHolder> {
    List<YouTubeVideos> youtubeVideoList;

    public VideoAdapter() {
    }

    public VideoAdapter(List<YouTubeVideos> youtubeVideoList) {
        this.youtubeVideoList = youtubeVideoList;
    }

    @Override
     public VideoViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {

     View view = LayoutInflater.from( parent.getContext())
                                    .inflate(R.layout.visor_video, parent,
false);

     return new VideoViewHolder(view);

    }

    @Override
    public void onBindViewHolder(VideoViewHolder holder, int position) {

        holder.videoWeb.loadData( youtubeVideoList.get(position)
                                                .getVideoUrl(), 'text/html'
, 'utf-8' );

    }

    @Override
    public int getItemCount() {
        return youtubeVideoList.size();
    }

    public class VideoViewHolder extends RecyclerView.ViewHolder{
```

```
        WebView videoWeb;

        public VideoViewHolder(View itemView) {
            super(itemView);

            videoWeb = (WebView) itemView.findViewById(R.id.wvVideo);

            videoWeb.getSettings().setJavaScriptEnabled(true);
            videoWeb.setWebChromeClient(new WebChromeClient() {
                } );
            }
        }
    }
```

7. Finalmente, invocamos a los videos desde el archivo **Main_Activity.java** colocando el siguiente código:

```
    package com.example.toshibapro.pmcaso102;

    import android.support.v7.app.AppCompatActivity;
    import android.os.Bundle;
    import android.support.v7.widget.LinearLayoutManager;
    import android.support.v7.widget.RecyclerView;

    import java.util.Vector;

    public class MainActivity extends AppCompatActivity {

        RecyclerView recyclerView;
        Vector<YouTubeVideos> youtubeVideos = new Vector<YouTubeVideos>();
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

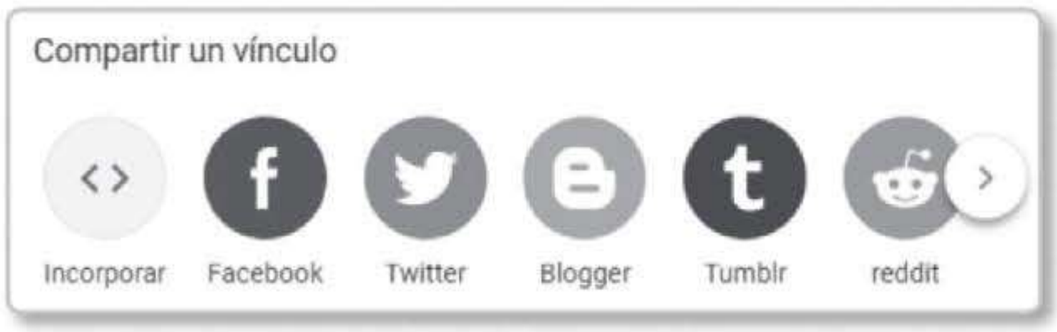
            recyclerView = (RecyclerView) findViewById(R.id.rvVideos);
            recyclerView.setHasFixedSize(true);
            recyclerView.setLayoutManager( new LinearLayoutManager(this));

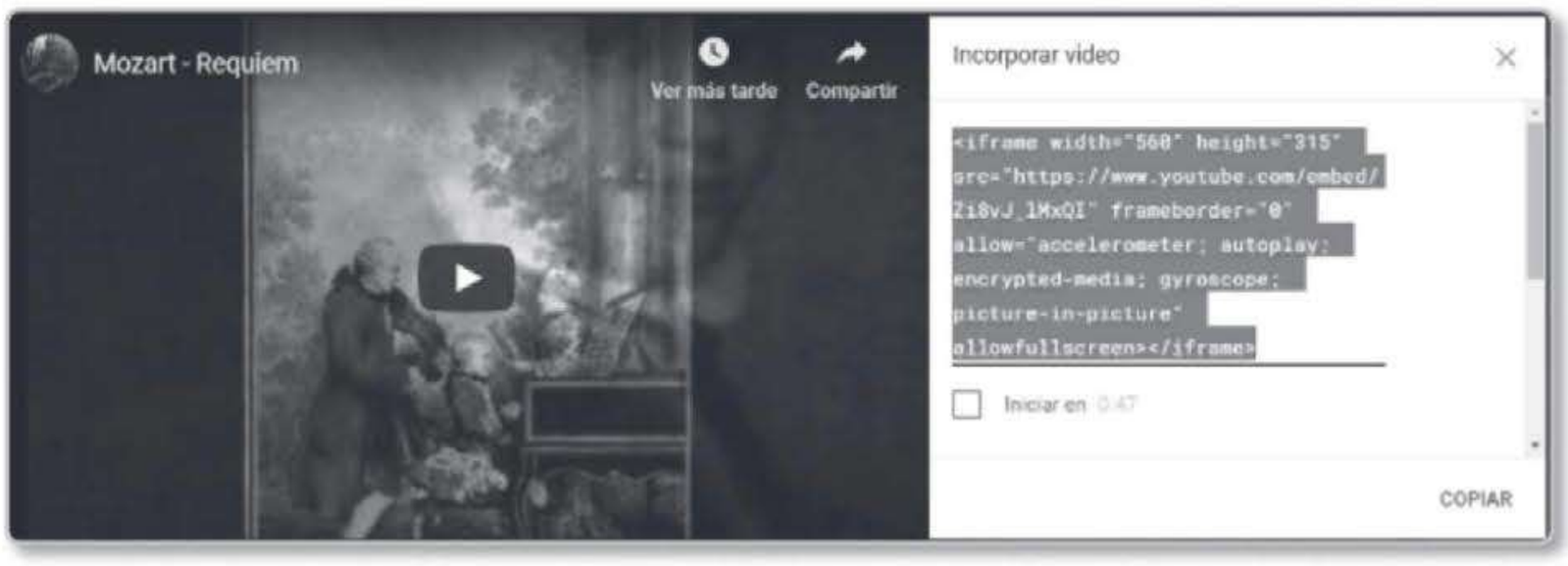
             youtubeVideos.add( new YouTubeVideos('<iframe width=\'100%\'
    height=\'100%\' src=\'https://www.youtube.com/embed/3cvmONlV5WU\'
    frameborder=\'0\' allow=\'accelerometer; autoplay; encrypted-media;
    gyroscope; picture-in-picture\' allowfullscreen></iframe></iframe>') );
             youtubeVideos.add( new YouTubeVideos('<iframe width=\'100%\'
    height=\'100%\' src=\'https://www.youtube.com/embed/SY3Kxf7ZTeI\'
    frameborder=\'0\' allow=\'accelerometer; autoplay; encrypted-media;
    gyroscope; picture-in-picture\' allowfullscreen></iframe>') );
             youtubeVideos.add( new YouTubeVideos('<iframe width=\'100%\'
    height=\'100%\' src=\'https://www.youtube.com/embed/Zi8vJ_lMxQI\'
    frameborder=\'0\' allow=\'accelerometer; autoplay; encrypted-media;
    gyroscope; picture-in-picture\' allowfullscreen></iframe>') );

            VideoAdapter videoAdapter = new VideoAdapter(youtubeVideos);

            recyclerView.setAdapter(videoAdapter);
        }
    }
```

Para obtener la dirección del video desde Youtube, seleccionamos **Compartir > Incorporar desde el mismo video**, tal como se muestra en la siguiente captura:





8. No olvidemos agregar las siguientes dependencias antes de iniciar la aplicación:

```
    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        implementation 'com.android.support:appcompat-v7:28.0.0'
        implementation 'com.android.support:recyclerview-v7:28.0.0'
        implementation 'com.android.support:cardview-v7:28.0.0'

       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'
    }
```

CASO DESARROLLADO 03: **visor de video desde Youtube usando credenciales**

Implementemos una aplicación móvil que permita mostrar un video desde Youtube usando credenciales.



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 67):

Tabla 67. Especificación de valores del proyecto

Create Android Project	Application Name : pmCaso103 Company Domain : Modifique por un nombre adecuado Project Location : C:\Libro Android\Capitulo_10
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)
Add an Activity to Mobile Configure Activity	Seleccionar : Empty Activity Activity Name : MainActivity Layout Name : activity_main

3. Modificamos el archivo **activity_main.xml** para colocar el visor de video y un botón que lo accionará, tal como se aprecia en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.youtube.player.YouTubePlayerView
        android:id="@+id/youtube_player"
        android:layout_height="wrap_content"
        android:layout_width="match_parent" />

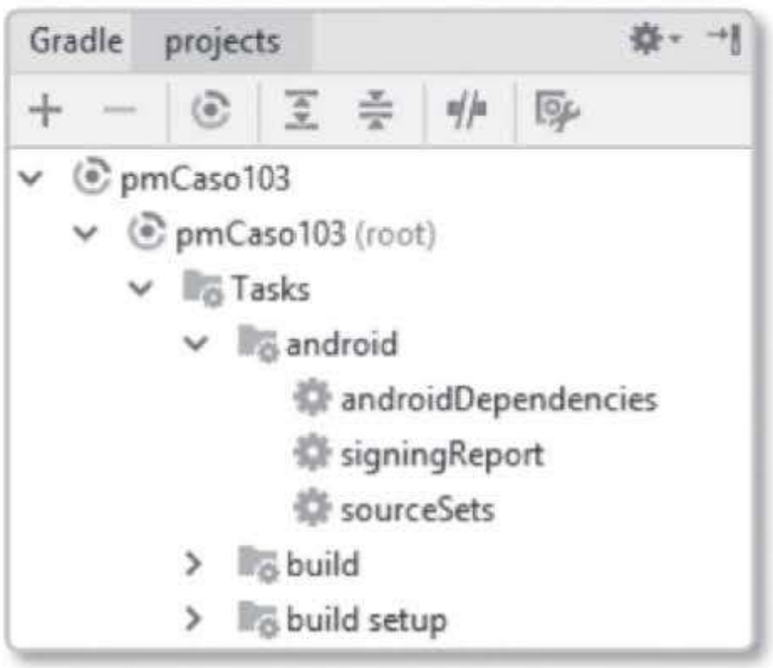
    <Button
        android:id="@+id/btMostrar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="MOSTRAR VIDEO" />

</RelativeLayout>
```

4. En el buscador Google, digitamos Google API Console; luego, seleccionamos Credenciales, y creamos una credencial para Youtube. Para ello, presionamos el botón Crear credenciales > Clave API > y asignamos un nombre a la clave. En nuestro caso, le asignamos YouTubeVideos. En restricción de aplicación, seleccionamos Aplicaciones de Android, y en restringir el uso a tus aplicaciones de Android, seleccionamos Añadir elemento. Aquí debemos copiar el nombre del paquete de nuestra aplicación y el codigo SHA-1 de esta.



Estos datos los podemos obtener desde el archivo **signingReport** de la pestaña **Gradle**, tal como se muestra a continuación.



Dentro del archivo se nos presenta los siguientes códigos. Seleccionamos solo el código de SHA1 y lo copiamos al nuevo elemento.

```
11:15:00: Executing task 'signingReport'...

Executing tasks: [signingReport]

:app:signingReport
Variant: debugAndroidTest
Config: debug
Store: C:\Users\TOSHIBA PRO\.android\debug.keystore
Alias: AndroidDebugKey
MD5: 87:C7:92:95:5F:B0:E1:40:FD:CC:CD:D9:6D:91:7F:03
SHA1: D8:AA:E3:88:D3:FD:04:7C:10:67:EC:91:9E:C5:B7:A7:A7:17:90:1A
Valid until: martes 25 de mayo de 2049
```

Al final, obtendremos una ventana parecida a la siguiente:



5. Agregamos la clase **PlayerConfig.java** al proyecto y colocamos el siguiente código:

```
package com.example.toshibapro.pmvideo1;

public class PlayerConfig {

    //Metodo constructor
    PlayerConfig(){
    }

    //Especificacion de la clave API
    public static final String API_KEY='AIzaSyDlZrnIV2Pe1EAgiG8zOyUZRV_
G7ApgwsE';
}
```

En cuanto a la variable **API_KEY**, la copiamos desde la ventana de credenciales.

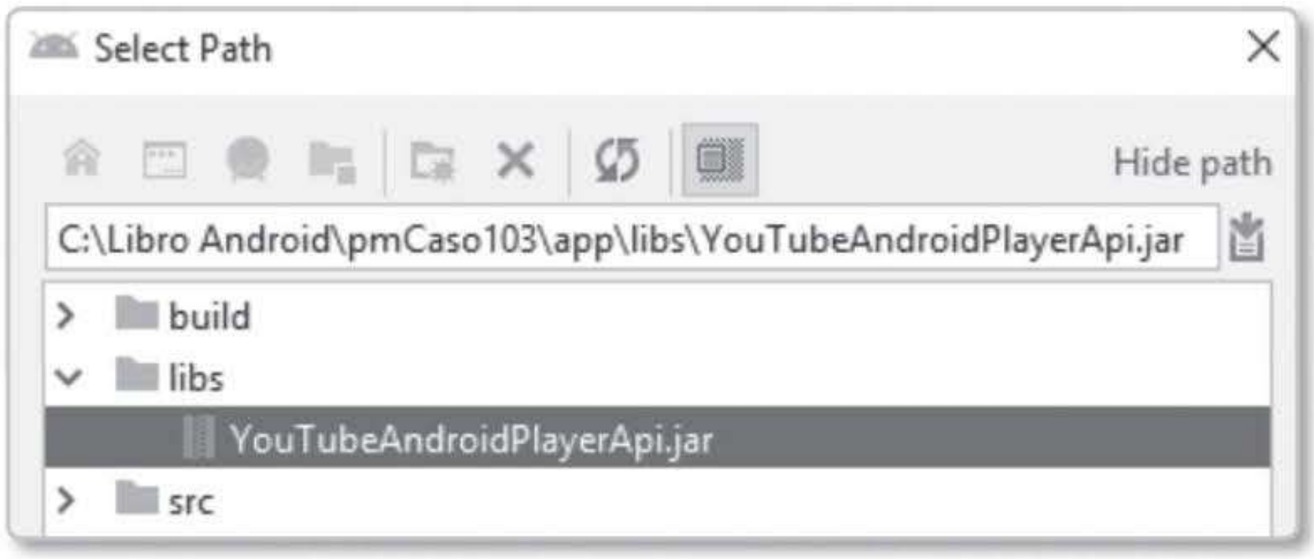
6. Agregamos el Youtube Android Player API desde la URL https://developers.google.com/youtube/android/player/downloads/ y el archivo.



Una vez descargado, copiamos el archivo **YouTubeAndroidPlayer.jar** dentro de la carpeta Libs de la aplicación, de tal forma que se muestre de la siguiente manera:



Para copiar, cambiamos la vista del explorador de **Android** a **Projects**; luego, debemos habilitarlo presionando clic derecho sobre el proyecto **> Open Module Settings > ficha Dependencias > Click en + > Jar Dependency >**



7. Modificamos la clase **Main_Activity.java** colocando el siguiente código:

```
package com.example.toshibapro.pmvideo1;

import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.MediaController;
import android.widget.VideoView;

import com.google.android.youtube.player.YouTubeBaseActivity;
import com.google.android.youtube.player.YouTubeInitializationResult;
import com.google.android.youtube.player.YouTubePlayer;
import com.google.android.youtube.player.YouTubePlayerView;

public class MainActivity extends YouTubeBaseActivity {

    YouTubePlayerView youTubePlayerView;
    Button btMostrar;
    YouTubePlayer.OnInitializedListener onInitializedListener;

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

        btMostrar = (Button) findViewById(R.id.btMostrar);
        youTubePlayerView = (YouTubePlayerView)findViewById(R.id.youtube_
player);
       onInitializedListener = new YouTubePlayer.OnInitializedListener() {
           @Override
            public void onInitializationSuccess(YouTubePlayer.Provider
provider, YouTubePlayer youTubePlayer, boolean b) {
               youTubePlayer.loadVideo('yAzx8vF4Bl8');
           }

           @Override
            public void onInitializationFailure(YouTubePlayer.Provider
provider, YouTubeInitializationResult youTubeInitializationResult) {

           }
       };
       btMostrar.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
                   youTubePlayerView.initialize(PlayerConfig.API_
KEY,onInitializedListener);
           }
       });
    }
}
```

8. Modificamos el archivo **AndroidManifest.xml** colocando el siguiente código:

```
<?xml version='1.0' encoding='utf-8'?>
<manifest xmlns:android='http://schemas.android.com/apk/res/android'
    package='com.example.toshibapro.pmvideo1'>

    <uses-permission android:name='android.permission.INTERNET'/>
     <uses-permission android:name='android.permission.ACCESS_NETWORK_
STATE'/>
    <uses-permission android:name='android.permission.ACCESS_WIFI_STATE'/>

    <application
        android:allowBackup='true'
        android:icon='@mipmap/ic_launcher'
        android:label='@string/app_name'
        android:roundIcon='@mipmap/ic_launcher_round'
        android:supportsRtl='true'
        android:theme='@style/AppTheme'>
        <activity android:name='.MainActivity'>
            <intent-filter>
                <action android:name='android.intent.action.MAIN' />

                <category android:name='android.intent.category.LAUNCHER' />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

CASO DESARROLLADO 04: **listado de audios**

Implementemos una aplicación móvil que permita reproducir una lista de audios mediante imágenes.



Solución

1. Seleccionamos **File>New>New Project**... desde Android Studio.
2. Asignamos los siguientes datos (ver Tabla 68):

Tabla 68. Especificación de valores del proyecto

Create Android Project	Application Name	: pmCaso104
	Company Domain	: Modifique por un nombre adecuado
	Project Location	: C:\Libro Android\Capitulo_10
Target Android Device	Activar: Phone and Tablet API 23: Android 6.0 (Marshmallow)	
Add an Activity to Mobile Configure Activity	Seleccionar	: Empty Activity
	Activity Name	: MainActivity
	Layout Name	: activity_main

3. Modificamos el archivo **activity_main.xml** para colocar el visor de video y un botón que lo accionará, tal como se observa en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:gravity="center"
        android:textStyle="bold"
        android:textColor="#000"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:text="LISTADO DE AUDIOS" />

    <ImageView
        android:id="@+id/ivAudio1"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        app:srcCompat="@drawable/music" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:gravity="center"
        android:text="Audio 01" />

    <ImageView
```

```
        android:id="@+id/ivAudio2"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        app:srcCompat="@drawable/music" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:gravity="center"
        android:text="Audio 02" />

    <ImageView
        android:id="@+id/ivAudio3"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        app:srcCompat="@drawable/music" />

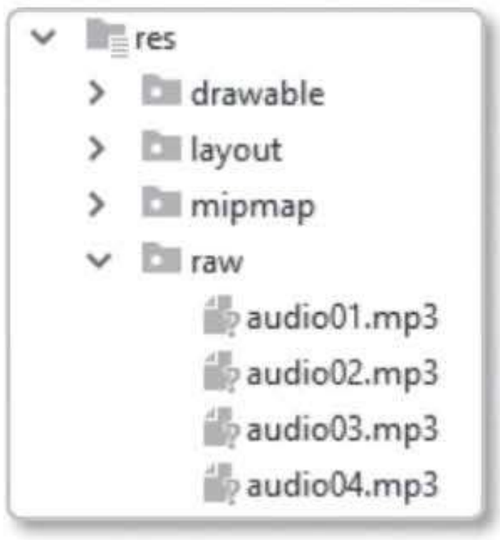
    <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:gravity="center"
        android:text="Audio 03" />

    <ImageView
        android:id="@+id/ivAudio4"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        app:srcCompat="@drawable/music" />

    <TextView
        android:id="@+id/textView5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        android:gravity="center"
        android:text="Audio 04" />

</LinearLayout>
```

4. Los audios que ejecutará la aplicación deberán colocarse dentro de la carpeta **raw** (clic derecho en **res > Android Resource Directory > Resource Type: raw**) y, luego, se debe copiar los audios:



5. Modificamos el archivo **MainActivity.java** para ejecutar los audios mediante las imágenes.

```
package com.example.toshibapro.pmcaso104;

import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity implements View.
OnClickListener {

    ImageView ivAudio1,ivAudio2,ivAudio3,ivAudio4;
    MediaPlayer mPlayer;

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

        ivAudio1 = (ImageView)findViewById(R.id.ivAudio1);
        ivAudio2 = (ImageView)findViewById(R.id.ivAudio2);
        ivAudio3 = (ImageView)findViewById(R.id.ivAudio3);
        ivAudio4 = (ImageView)findViewById(R.id.ivAudio4);

        ivAudio1.setOnClickListener(this);
        ivAudio2.setOnClickListener(this);
        ivAudio3.setOnClickListener(this);
        ivAudio4.setOnClickListener(this);
    }

    public void onClick(View v){
        //Liberar al Media Player
        if (mPlayer != null) mPlayer.release();
        int id = v.getId();

        if (id == R.id.ivAudio1){
            mPlayer = MediaPlayer.create(this, R.raw.audio01);
        }

        if (id == R.id.ivAudio2)
            mPlayer = MediaPlayer.create(this, R.raw.audio02);
        if (id == R.id.ivAudio3)
            mPlayer = MediaPlayer.create(this, R.raw.audio03);
        if (id == R.id.ivAudio4)
            mPlayer = MediaPlayer.create(this, R.raw.audio04);

            mPlayer.seekTo(0);

            mPlayer.start();
        }

        public void onPause(){
            super.onPause();
            if (mPlayer != null) mPlayer.release();
        }
    }
```

CAPÍTULO

11

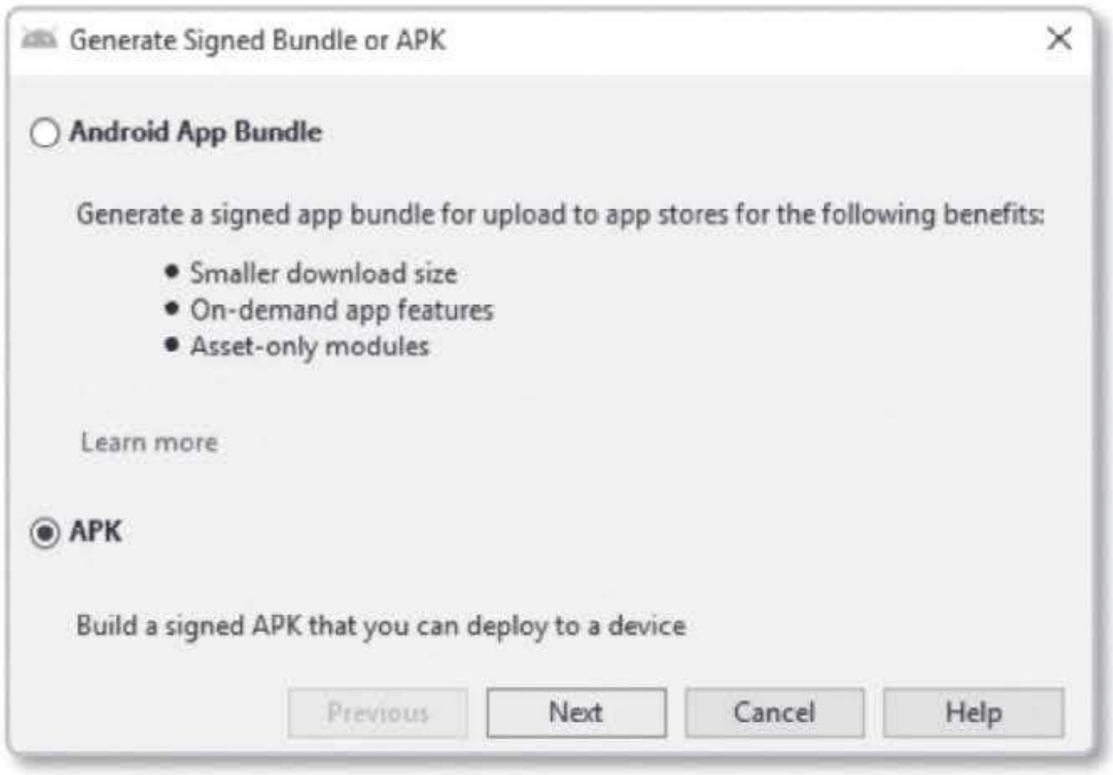


Publicación en Google Play

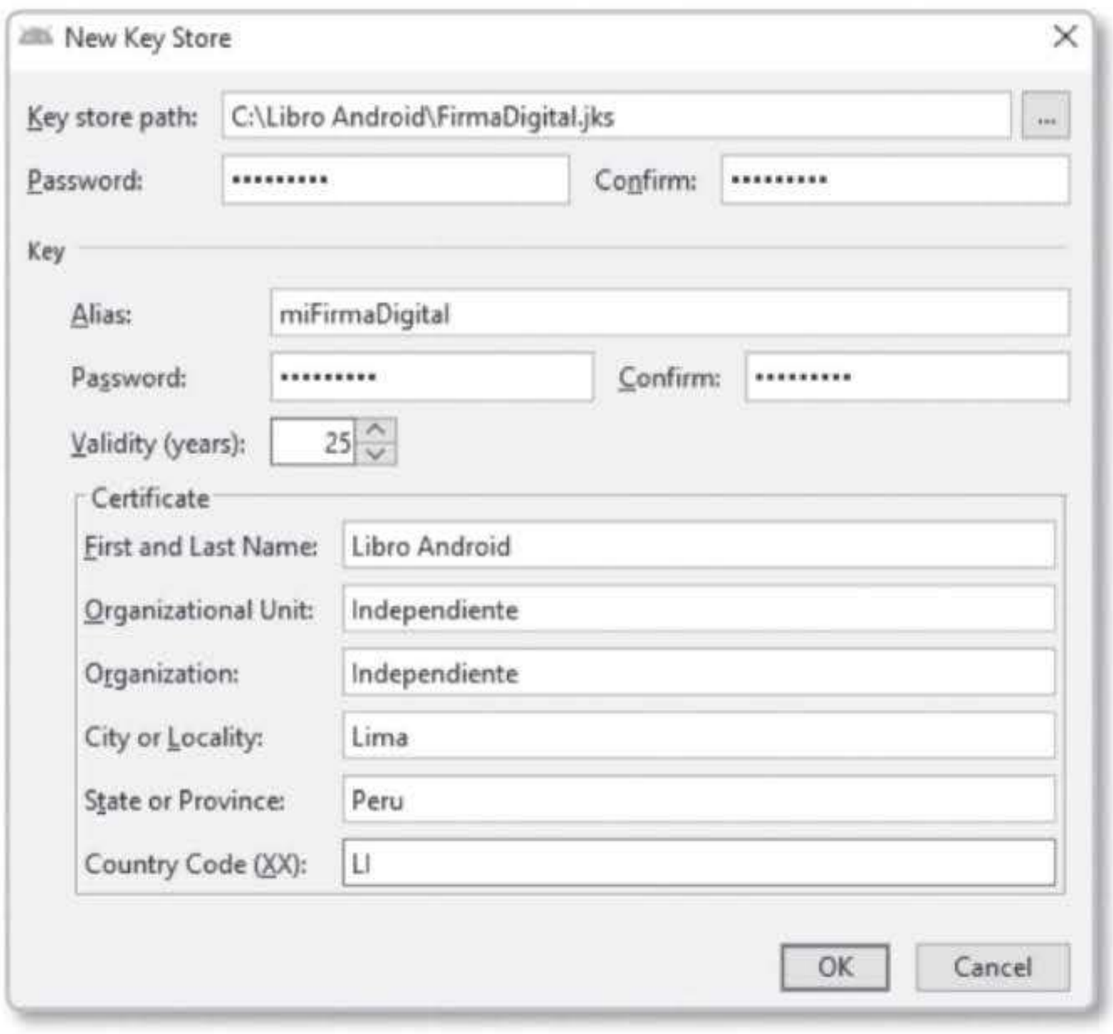
11.1 Generación de APK con certificado digital

Para publicar nuestras aplicaciones Android, debemos crear el archivo de ejecución llamado APK. Estos archivos serán enviados a los dispositivos móviles, serán ejecutados como fuentes desconocidas y probados las aplicaciones desde su dispositivo. Para esto, antes debemos considerar las siguientes especificaciones:

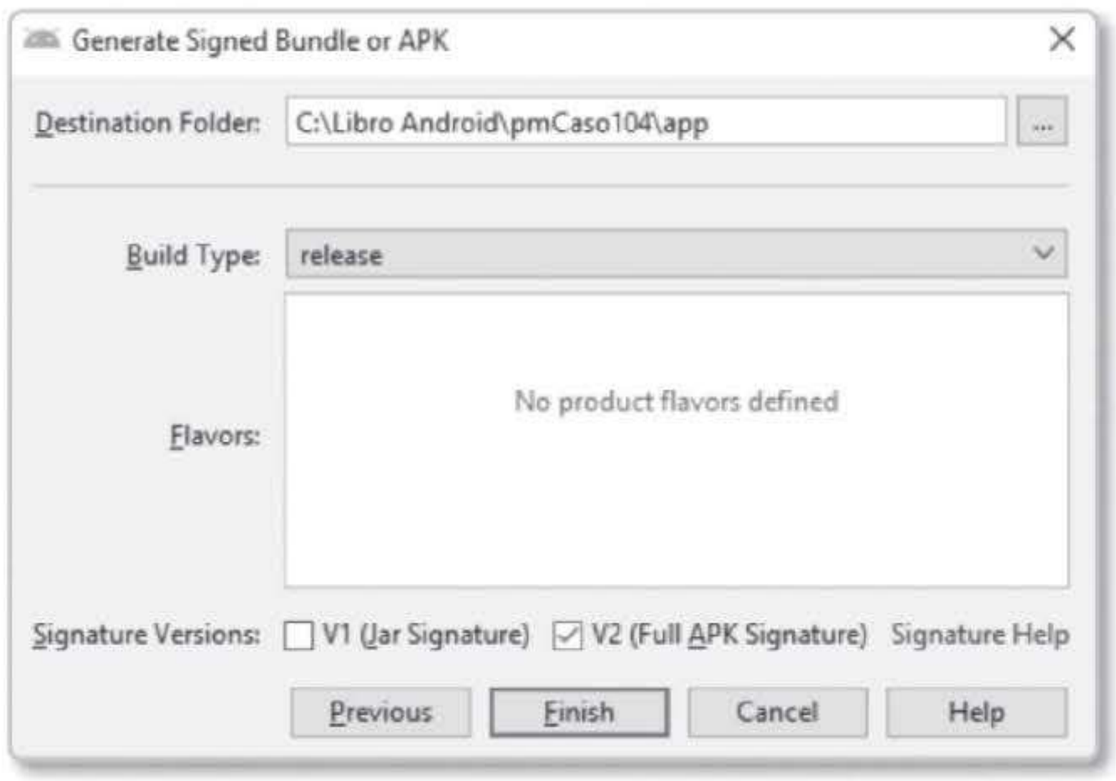
- La aplicación debe contar con un nombre, ícono y el número de versión. Desde Android Studio seleccionamos **Build > Generate Signed Bundle or APK**.



De la ventana anterior, seleccionamos **APK** y haga clic en **Siguiente > Create new**... Seguidamente, llenamos toda la información concerniente a la aplicación, tal como se muestra en la siguiente imagen:



- La aplicación no debe contener mensajes de error.
- Probamos la aplicación y nos aseguramos de que se ejecute correctamente.
- Luego, seleccionamos **V2 (Full APK Signature)**, tal como se muestra en la siguiente captura:



- Finalmente, creamos el APK con la tecla "Finish".

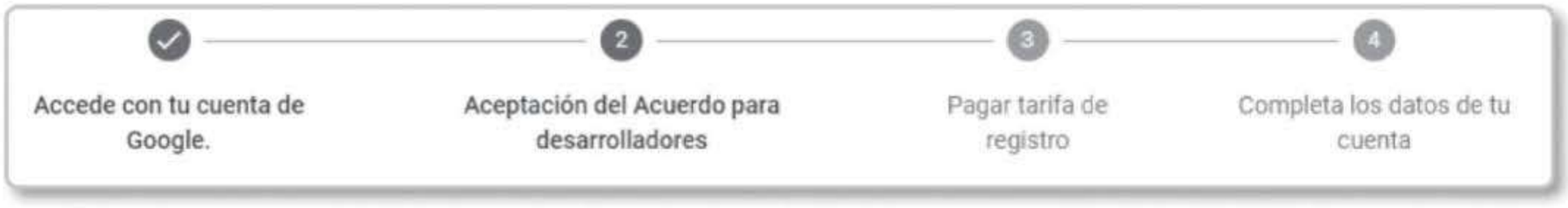
11.2 Creación de una cuenta en Google Play

Una vez que tenemos una aplicación Android probada, existe la necesidad de compartir o distribuirla. El único medio para hacerlo es la tienda virtual Android (ver Figura 36).



Figura 36. Play Store Android. Tomado de "7 trucos que desconoces de la tienda de Android Play Store," por A. Bret, 2017. Recuperado de https://www.tuexpertoapps.com/2017/06/07/7-trucos-que-desconoces-de-la-tienda-de-android-play-store/.

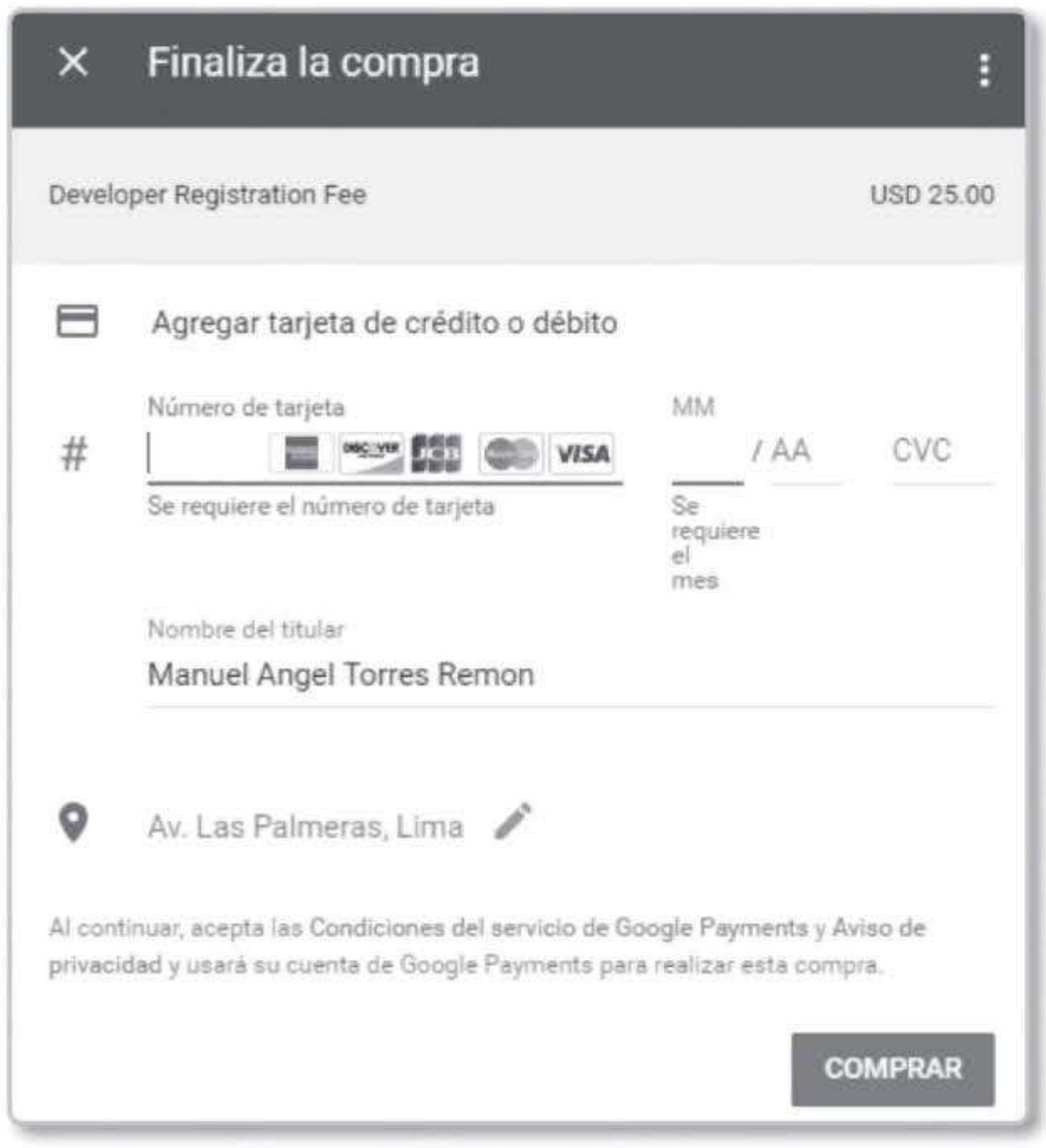
Debemos seguir cuatro pasos principales, tal como se observa en la siguiente imagen:



Para iniciar con el proceso, accedemos al siguiente URL:https://play.google.com/apps/publish/signup/. Antes de iniciar la publicación de aplicaciones en la tienda, debemos conocer que tiene un costo de US$25 únicos, que se realiza completando los siguientes pasos:



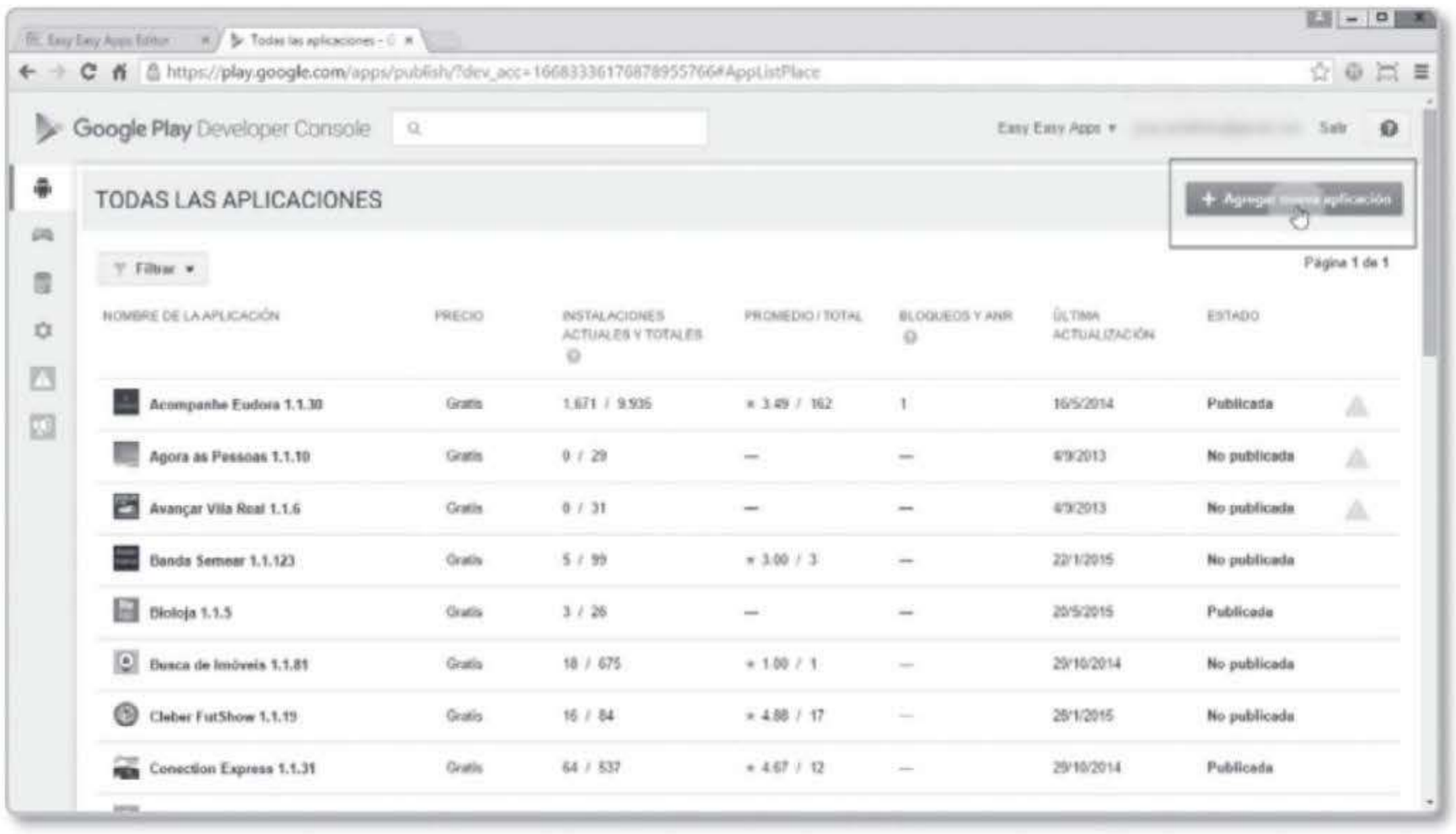
Finalmente, se debe realizar el pago mediante la siguiente ventana:



11.3 Publicación de la aplicación

Una vez gestionado el pago por publicación, podremos publicar nuestras aplicaciones entrando a la URL: **https://play.google.com/apps/publish/.**

- Seleccionamos el botón **+ Agregar nueva aplicación**, como se muestra en la siguiente ventana:



- Agregamos el idioma y el nombre de la aplicación. Seguidamente, seleccionamos el botón **Cargar archivo APK**, tal como se muestra en la siguiente ventana:



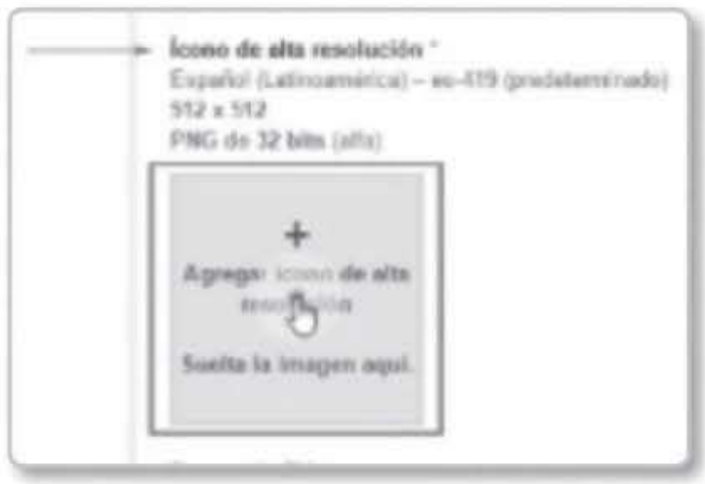
- Después, presionamos el botón **Subir el primer APK a producción**; luego, arrastramos el archivo APK desde la computadora a la ventana o también lo buscamos con el botón **Examinar archivos**.
- Luego, nos ubicamos en la pestaña **Ficha de Play Store**, y llenamos la información necesaria para la aplicación, la cual se podrá visualizar en la página de Google Play por los usuarios que la descargarán.



- Cargamos las imágenes que se visualizarán en nuestra aplicación (captura de pantalla) para mostrar a los usuarios. Al menos, debemos añadir dos imágenes, cuyos tamaños deben ser de al menos 320 píxeles de ancho.



- Debemos añadir un ícono a la aplicación con 512 x 512 píxeles con formato PNG de 32 bits. Este ícono es el que se presentará en el Google Play y en búsquedas de aplicaciones.



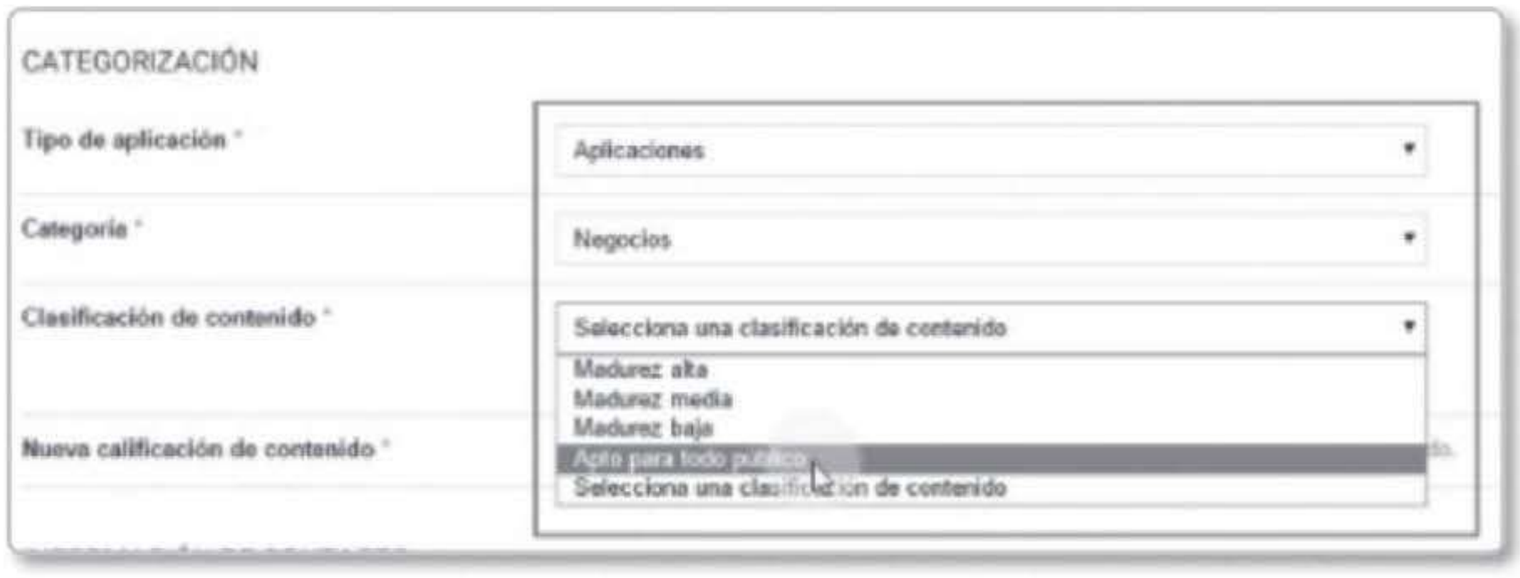
- Debemos añadir un gráfico de la función, el cual será visible en la parte superior de la página de aplicaciones de Google Play. Esta imagen debe ser de 1024 x 500 píxeles, y debe estar en formato JPG o PNG.



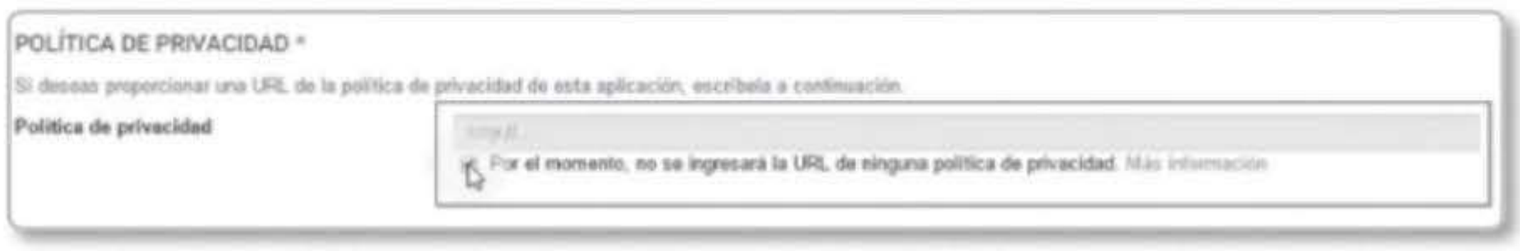
- En el campo Video promocional, podemos incluir un video desde Youtube, con el cual se muestre las cualidades de nuestra aplicación. Esta sección es opcional.



- En Categorización, especificaremos algunas características de la aplicación, por ejemplo, el tipo "Aplicaciones", la categoría Educación o Negocio, y la clasificación de contenidos, por ejemplo, indicaremos que es apta para todo público.



- En Información de Contacto, agregamos información referente a la empresa o propietario de la aplicación para algún contacto.
- En Política de Privacidad, podemos direccionar a un sitio web que contenga información sobre las políticas de uso de la aplicación.



- Luego, debemos seleccionar **Guardar Borrador** para continuar con el proceso de publicación.
- Luego, respondemos un cuestionario sobre la publicación que se está realizando. Al finalizar el cuestionario, seleccionamos el botón **Aplicar Clasificación**.
- Luego, definiremos el precio y cómo se distribuirá la aplicación; además, podemos seleccionar los países donde esta estará disponible.



- Finalmente, seleccionamos el botón **Publicar aplicación** y esperamos un correo electrónico de confirmación.

Referencias

Digital

Agarwal, N. (2018). *Android View Classes*. Recuperado de https://www.formget.com/android-views/

Android Concepts. (2014). Android Intent Service. Recuperado de http://androidconceptspp.blogspot.com/2014/06/android-intent-service.html.

Aprende y comparte. Tema fantástico. (2013). Conceptos Generales Android. Archivo AndroidManifest.xml. Recuperado de http://androiddummies.blogspot.com/2013/06/conceptos-generales-android-archivo.html

Bret, A. (2017). 7 *trucos que desconoces de la tienda de Android Play Store*. Recuperado de https://www.tuexpertoapps.com/2017/06/07/7-trucos-que-desconoces-de-la-tienda-de-android-play-store/

Codepath. (s.f.). *Handling ProgressBars*. Recuperado de https://guides.codepath.com/android/handling-progressbars.

Desarrollador Android. (2015). *Soporte de diferentes pantallas*. Recuperado de https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/soporte-de-diferentes-dispositivos/soporte-de-diferentes-pantallas/

Developers (s.f.). *Intents y filtros de intents*. Recuperado de https://developer.android.com/guide/components/intents-filters?hl=es-419

Developers. (s.f.). *Fragmentos*. Recuperado de https://developer.android.com/guide/components/fragments?hl=es-419

Digital Learning SL. (s.f.). *Palette.pngw 806 x 622 pixeles*. Recuperado de https://academiaandroid.com/wpaa/wp-content/uploads/2016/05/Palette.png

Digital Learning SL. (s.f.). *Elementos y diseño de Interfaz de Usuario en Apps Android*. Recuperado de https://academiaandroid.com/elementos-y-diseno-de-interfaz-de-usuario-en-apps-android/

El Androide. (s.f.). *Qué es acelerómetro en un móvil y para qué sirve*. Recuperado de https://elpoderdelandroideverde.com/que-es-acelerometro-en-un-movil/

Fox, J. (2017). *The internet of things: A beginner's guide to commercial applications and security risks*. Recuperado de https://www.ifsecglobal.com/global/the-internet-of-things-a-beginners-guide-to-commercial-applications-and-security-risks/

Henao, C. (2017, 28 de junio). Personalizar Navigation Drawer - Iconos y Banne [video]. Recuperado de https://www.youtube.com/watch?v=s-inDBRyboA

o7planning. (s.f.). 1- What is ImageView?. Recuperado de https://o7planning.org/en/10531/android-imageview-tutorial

Palacios, J. (2018). *Google Maps cambia su indicador por carritos para guiarte*. Recuperado de https://www.sopitas.com/entretenimiento/google-maps-autos-colores-actualizacion/

Paneva, S. (2018). How to make your Mobile Application GDPR-Ready. Recuperado de https://blog.sashido.io/how-to-make-your-mobile-application-gdpr-ready/

Ramírez, I. (2017). *¿Cuál es el mejor reproductor de vídeo para Android?* Analizamos a fondo los siete mejores. Recuperado de https://www.xatakandroid.com/

Revelo, J. (2015). *Actividades en Android: concepto y ciclo de vida*. Recuperado de http://www.hermosaprogramacion.com/2018/09/actividades-ciclo-de-vida/

Simpson, R. (2017). *Android overtakes Windows for first time*. Recuperado de http://gs.statcounter.com/press/android-overtakes-windows-for-first-time

Stackoverflow. (2013). *How can I make the spinner width same as largest available item width*. Recuperado de https://stackoverflow.com/questions/16633705/how-can-i-make-the-spinner-width-same-as-largest-available-item-width

Stackoverflow. (2015). *How can I change default black dim background "color" (not the amount of dim) of Dialog?* Recuperado de https://stackoverflow.com/questions/29227126/how-can-i-change-default-black-dim-background-color-not-the-amount-of-dim-of/29482234

Tarek, A. (2018). *Android Custom View Tutorial*. Recuperado de https://www.raywenderlich.com/142-android-custom-view-tutorial

The Phone House Spain. (2017). Qué es la densidad de píxeles y en qué te afecta. Recuperado de https://blog.phonehouse.es/2017/02/01/que-es-la-densidad-de-pixeles-y-en-que-te-afecta/

Tijero, E. (2016). El ciclo de vida del Activity en Android. Recuperado de https://stories.devacademy.la/android-ciclo-de-vida-de-un-activity-b2a68c506961

Time Timer. (s.f.). *Time Timer® Android App*. Recuperado de https://www.timetimer.com/products/time-timer-android-app.

Tripathi, A. (2017). *Android Beginners : Views & Layouts*. Recuperado de https://medium.com/@geekanamika/android-beginners-views-layouts-657a5bbeebe2

TutorialPoints. (s.f.). *Android-Services*. Recuperado de https://www.tutorialspoint.com/android/android_services.htm

Verlayne RGL. (2019). *¿Cómo controlar la orientación vertical en aplicaciones?* Recuperado de https://galaxyandroid.com/controlar-orientacion-vertical/.

Weweb. (s.f.). *El uso de dispositivos móviles crece muy rápidamente. La importancia de los usuarios*. Recuperado de http://www.websresponsivas.com/

Libros

Ribas, J. (2017). *Desarrollo de aplicaciones para Android*. Ciudad, Estado de Estados Unidos: Anaya.

Phillips, B., Stewart, C., Hardy, B., & Marsicano, K. (2016). *Programación en Android*. Ciudad, Estado de Estados Unidos: Anaya.

Impreso en los talleres gráficos de



EDITORIAL
MACRO®

Surquillo

Julio 2019