

ANDROID STUDIO

Aprende a desarrollar aplicaciones



José Dimas Luján Castillo

Android Studio

Aprende a desarrollar aplicaciones

José Dimas Luján Castillo



Diseño de colección y preimpresión:
Grupo RC
Diseño cubierta: Cuadratín

Datos catalográficos

Luján, José Dimas
Android Studio. Aprende a desarrollar aplicaciones
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-538-009-4

Formato: 17 x 23 cm

Páginas: 332

Android Studio. Aprende a desarrollar aplicaciones

José Dimas Luján Castillo

ISBN: 978-84-947170-1-7 edición original publicada por RC Libros, Madrid, España.

Derechos reservados © 2017 RC Libros

Primera edición: Alfaomega Grupo Editor, México, septiembre 2017

© 2018 Alfaomega Grupo Editor, S.A. de C.V.

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720, Ciudad de México.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro No. 2317

Pág. Web: <http://www.alfaomega.com.mx>

E-mail: atencionalcliente@alfaomega.com.mx

ISBN: 978-607-538-009-4

Derechos reservados:

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México – Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

ÍNDICE

Prefacio	XI
Capítulo 1. Introducción	1
Acerca de este libro.....	2
Capítulo 2. ¿Qué es Android?	3
Historia de Android	3
Versiones de Android	3
Estructura del sistema operativo	4
Linux Kernel (Capa ROJA)	5
Libraries (Capa VERDE).....	5
Entorno de ejecución (Capa AMARILLA)	5
Framework de aplicaciones (Capa AZUL)	6
Aplicaciones (Capa AZUL)	6
Capítulo 3. Android Studio	7
Netbeans	7
Eclipse	7
Android Studio	8
¿Qué se necesitará para desarrollar en Android?	8
Obteniendo Java	8
Android Studio	10
Configurar Android Studio para programar	14
Emulador.....	17
Configurar un emulador	19

Capítulo 4. Crear una aplicación	25
Crear una aplicación.....	25
Capítulo 5. Estructura de un proyecto	31
Perspectivas	32
Estructura de un proyecto.....	32
Directorio manifests.....	32
Directorio Java.....	36
Directorio res	38
drawable	39
layout	39
mipmap	40
values	40
Gradle.....	41
Gradle Scripts	41
build.gradle (Project).....	42
build.gradle (Module)	43
Capítulo 6. Activity.....	45
Ciclo de vida de una activity	47
Capítulo 7. Layouts	51
Tipos de layouts	53
LinearLayout.....	54
RelativeLayout.....	55
AbsoluteLayout	56
TableLayout.....	57
FrameLayout	58
Capítulo 8. Elementos gráficos en Android	61
Interfaz gráfica	63
Pestaña “Design”	64
Pestaña “Text”	64
TextView	67

Probando la orientación LinearLayout	68
Propiedades del TextView.....	73
Button	76
EditText	78
Practicando RelativeLayout.....	80
Margin y Padding	88
Pesos y distribución.....	96
Capítulo 9. Densidades y tamaños de pantalla	103
Densidades.....	103
Tamaños de pantalla	104
Relación densidad y tamaño de pantalla	104
Capítulo 10. Medidas en Android	107
Píxeles (PX)	107
Pulgada (inch).....	108
Milímetros (mm)	108
Puntos (pt)	109
Píxeles de densidad independiente (dp)	109
Capítulo 11. Pantallas para múltiples dispositivos	111
Orientación de pantalla.....	111
Portrait	112
Landscape	115
Forzar solamente una opción de orientación.....	124
Tamaños de pantallas	126
Capítulo 12. Archivo String	133
Capítulo 13. Imágenes.....	139
Características de imágenes.....	139
ImageView	140
Directorios drawable.....	142
Escala de imágenes	145

Capítulo 14. Interacción en una aplicación	149
Java, Programación orientada a objetos	150
Clase.....	150
Objeto	152
Herencia.....	153
Modificadores de acceso.....	154
Acceder a elementos desde Java	155
Toast.....	163
Capítulo 15. Agregar una Activity	169
Invocar una Activity.....	173
Intent.....	176
Intent para llamadas	179
Permisos para llamar.....	182
Intent para enviar SMS.....	184
Capítulo 16. WebView	189
Capítulo 17. Audio	195
MediaPlayer	195
Directorio raw	196
Capítulo 18. Sensores.....	203
Capítulo 19. Fragmentos	211
Creando la interfaz gráfica delos fragmentos	213
Creando la interfaz de comunicación	214
Creando código de los fragmentos.....	215
Registrar los fragmentos en el manifest.....	221
Capítulo 20. Pérdida de información	225
Evitar pérdida de datos	232
Capítulo 21. ListView	237
Capítulo 22. RecyclerView.....	251
LayoutManager	259

Adapter	260
Comparando RecyclerView y ListView	267
Capítulo 23. CardView.....	269
Capítulo 24. FloatingActionButton	279
Sentido de usar el FloatingActionButton.....	288
Capítulo 25. SnackBar	289
Acciones en el SnackBar	294
Diferencias entre SnackBar y Toast.....	301
Capítulo 26. Publicar una aplicación	303
Generar APK.....	305
Subir APK a la Play Store	309
Ficha Play Store	311
Precio y distribución.....	315
Productos de compra en aplicaciones.....	316
Recomendaciones al publicar una aplicación.....	316
Índice Analítico	319

PREFACIO

Desde los inicios de las computadoras se ha tenido la necesidad de interactuar con estos dispositivos tecnológicos, con el paso de los años se logra una mayor comunicación, al grado que actualmente se pueden utilizar gestos, voz, movimientos y casi cualquier acción que nuestra imaginación nos permita pensar, todo esto con el fin de lograr una respuesta o ejecutar una acción en los dispositivos que conocemos como computadoras.

Una de las revoluciones tecnológicas más importantes en los últimos años es la de los dispositivos móviles, ya que han logrado comunicar a una persona con otra en el otro lado del mundo en un instante, esto produce una sensación de “tiempo real” ya que solo basta con el acceso a una línea telefónica, plan de datos o cualquier otro medio compatible con esta tecnología hoy en día.

En la actualidad el sistema operativo más usado dentro de los dispositivos móviles es sin duda Android. Este sistema operativo acapara según la mayoría de las estadísticas el 85%-90% de la cuota del mercado. ¿Te imaginas tener acceso a esos usuarios?

Android es un sistema operativo móvil que utiliza por detrás a un viejo conocido de la industria del software, el lenguaje de programación Java, aunque Android cuenta con sus propias clases, librerías, framework y otras herramientas. Algo recomendable para comenzar en el mundo del desarrollo móvil es informarse y estudiar Java, ya que este lenguaje cuenta con algunas reglas propias en comparación con otros lenguajes de programación.

El último gran avance en el mundo del desarrollo de aplicaciones Android es la herramienta “*Android Studio*”. Fue desarrollada por la misma compañía que está detrás de Android, Google.

Una forma de apoyar y volver más ágil el desarrollo de aplicaciones es utilizar *Android Studio*, esta herramienta vino a ser un punto de apoyo bastante sólido durante el desarrollo de aplicaciones, ya que cubrió algunas de las carencias que se

tenían con las herramientas de programación y, además, cambió ciertas reglas del desarrollo en Android.

Acerca del autor

José Dimas Luján Castillo nació en 1986 en la ciudad de México, en referencia a sus estudios el último grado que obtuvo es una Maestría en Tecnologías de Información en el ITESM.

En su experiencia como desarrollador tiene ya más de 200 aplicaciones realizadas para clientes, plataformas en línea, cursos y ejemplos en internet. Cuenta con la impartición de clases en más de 18 universidades a nivel presencial en Latinoamérica en los niveles de Licenciatura y Maestría.

Actualmente también es colaborador de las plataformas más importantes de educación en línea: Video2brain/ LinkedIn(Microsoft) y Escuela.it, codigofacilito. Cuenta con más de 170 cursos en línea en los últimos 5 años.

El autor proporciona su sitio web, redes sociales, twitter y correo a continuación:

Sitio web: www.josedlujan.com

Facebook: www.facebook.com/josedlujan

Twitter: www.twitter.com/josedlujan

Correo: josedlujan@gmail.com

El código de los ejercicios estará colgado en el repositorio del autor en GitHub: <https://github.com/josedlujan>.

Agradecimientos

Agradezco a mi pareja Noemí por el camino que estamos iniciando y por su apoyo incondicional en estos últimos años, además de apoyarme con la revisión en los textos de este libro. También a mis padres, Fabiola y José, que espero que sientan que el camino que iniciamos juntos hace 31 años ha valido mucho la pena, les agradezco su apoyo para siempre.

1 INTRODUCCIÓN

Android es un sistema operativo, no es lenguaje de programación, esto nos debe hacer conscientes de que desarrollar una aplicación móvil no solo involucra los conceptos que nos encontramos habitualmente en la programación como: variables, funciones, ciclos, condiciones. El desarrollo móvil implica que conozcamos como funciona un sistema operativo, permisos, implementación de algunos sensores, manejo de tareas, manejo de memoria y de procesos, entre otros. Podemos decir que la complejidad es mayor si la desarrollamos en un solo lenguaje de programación.

El reto del desarrollador hoy en día es entender el sistema operativo, interactuar con el usuario, con otras aplicaciones, con servidores y con tecnología que forma parte del mundo móvil.

En el mercado vamos a encontrar muchos sistemas operativos como: iOS, Firefox OS, BlackBerry, Windows Phone, Ubuntu, entre otros. Android cuenta con la mayor cuota de mercado, pero también con la mayor cuota de fabricantes, esto añade complejidad ya que no todo el hardware interactúa de la misma forma. Además, el fabricante es el encargado de proporcionar la actualización al usuario y nos encontramos con el problema de la fragmentación que eleva la complejidad para el desarrollador de aplicación.

Los párrafos anteriores tratan de describir la situación actual de un desarrollador Android, se busca que el lector sea consciente del reto e implicaciones a las que se va a enfrentar en el mundo del desarrollo, pero como todo esfuerzo después de haber trabajado, estudiado y practicado, te darás cuenta de lo interesante y productivo que puede ser desarrollar aplicaciones para el sistema operativo número uno del mercado.

Acerca de este libro

Este libro está dirigido a los desarrolladores novatos interesados en dar los primeros pasos en el mundo del desarrollo móvil Android; por tanto, no son necesarios los conocimientos en el lenguaje de programación Java, pero si se tienen conocimientos básicos de programación, y también de Java, podemos decir que va a ser mucho más fácil asimilar el contenido, y con las horas adecuadas dedicadas a su lectura y estudio resultará una herramienta de gran ayuda para crear aplicaciones Android.

Android como ya comenté en el inicio del capítulo, no es un lenguaje de programación como Java, Python y otros. Se recomienda tener paciencia y perseverancia para lograr el objetivo de desarrollar una aplicación.

El libro contiene impresiones de pantalla, y con ello se busca que el lector pueda entender con el apoyo visual lo que se está buscando con el ejemplo o ejercicio que se esté realizando.

¿QUÉ ES ANDROID?

Android es un sistema operativo al igual que Windows, Linux, etc. Por lo tanto, tiene el control total del dispositivo que lo contiene, así que cuando desarrollamos una aplicación, estamos desarrollando para el sistema operativo y podemos tener control de ciertos elementos que generar una experiencia más interesante y agradable para el usuario si lo hacemos de buena forma, en caso contrario podríamos estar generando una de las peores experiencias como usuario y afectando el rendimiento del dispositivo.

Historia de Android

Android en la actualidad pertenece a una de las compañías pilares del mundo de la tecnología, Google. Se sabe que los creadores fueron una pequeña empresa con el nombre de Android Inc., que fue creada en el año 2003 y un par de años atrás Google la compró con la intención de adquirir el proyecto estrella de la compañía Android, que era un sistema operativo para dispositivos móviles.

Google continuó con el reto de desarrollar un sistema operativo para dispositivos móviles, pero no lo hizo solo, para ello fundó “OHA” (*Handset Alliance*) en el año 2007 teniendo dentro de sus filas compañías tecnológicas gigantes y con renombre como: HTC, Texas Instruments, Motorola y obviamente a Google.

Después de esta colaboración y arduo trabajo se lanza Android en el mes de septiembre del año 2008, fecha oficial de su lanzamiento.

Versiones de Android

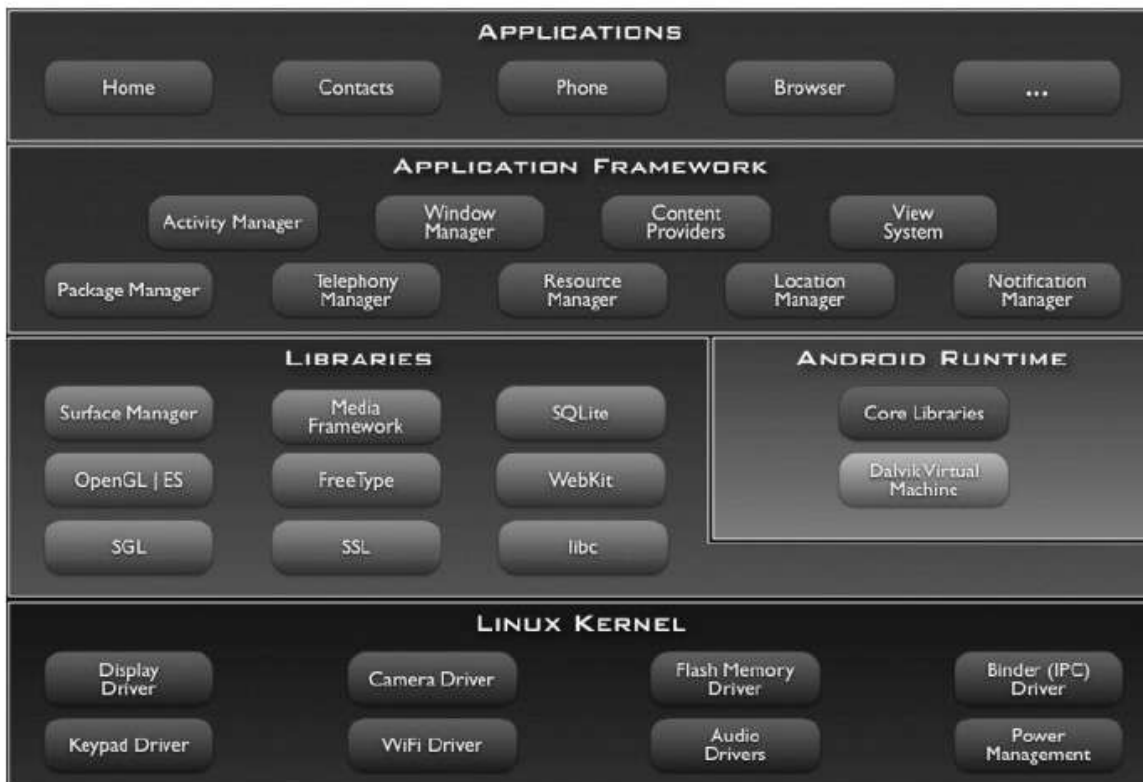
Los nombres de las versiones de Android siempre están en orden alfabético, es decir, que la primera versión se inició con la letra “A”, la segunda con la “B” y así

sucesivamente; el nombre siempre hace referencia a un postre en el idioma inglés. Hasta el día de hoy estas son las versiones disponibles:

- A - Apple Pie. Versión 1.0
- B - Banana Bread. Versión V 1.1
- C - Cupcake. Versión V1.5
- D - Donut. Versión V1.6
- E - Éclair. Versión V2.0
- F - Froyo. Versión V2.2
- G - Gingerbread. Versión V2.3
- H - Honeycomb. Versión V3.0
- I - Icecream Sandwich. Versión V4.0
- J - Jelly Bean. Versión V4.1
- K - KitKat. Versión V4.4
- L - Lollipop. Versión V5.0
- M – Marshmallow. Versión V6.0
- N – Nougat. Versión V7.0

Estructura del sistema operativo

La siguiente imagen nos ayudará a representar las capas del sistema operativo:



Linux Kernel (Capa ROJA)

El núcleo de Android es Linux, para ser específico hablamos del kernel de Linux versión 2.6, aunque se le agregaron partes necesarias para el hardware de los dispositivos móviles.

El desarrollador tiene muy poco acceso a esta capa, como desarrolladores notaremos la casi nula interacción, el desarrollador no se preocupa por los modelos de cámara, por ejemplo, solamente decimos y damos la orden a la cámara y esta capa nos resuelve la vida. Podemos decir que esta capa contiene los drivers o las instrucciones para la interacción con el hardware de los dispositivos.

Libraries (Capa VERDE)

Estas librerías están en los lenguajes C y C++, en su mayoría los fabricantes de los dispositivos se hacen cargo de estas capas, cada una de las librerías tiene sus funciones y las describimos a continuación:

- Surface Manager: gestiona el acceso a la pantalla.
- Media Framework imágenes, audio y vídeo.
- SQLite.
- WebKit.
- SGL.
- OpenGL.
- Freetype vectores o imágenes.

Entorno de ejecución (Capa AMARILLA)

No lo podemos considerar una capa, ya que internamente tiene librerías. Las librerías están divididas en dos: las de Java y las de Android.

En esta capa se encuentra una de las partes más conocidas de Android, la “Máquina Virtual”, durante mucho tiempo la máquina virtual fue Dalvik, pero en las últimas versiones de Android se cambió por ARM. Aunque en la imagen aparece Dalvik en los dispositivos actualizados debes de encontrar ARM como máquina virtual, esto para el usuario final pasa inadvertido.

La máquina virtual de Java es diferente a la máquina virtual de Android, por lo tanto los bytecode no son compatibles.

Framework de aplicaciones (Capa AZUL)

El desarrollador puede acceder a la mayor parte de esta capa sin problemas, aquí es donde encontramos la mayoría de las librerías que se usan en el día a día del desarrollador Android; describiremos brevemente cada una de las partes:

- Activity Manager: administra las actividades de nuestra aplicación y el ciclo de vida.
- Windows Manager: administra lo que se muestra en la pantalla.
- Content Provider: dependiendo de cómo le indiquemos algunos contenidos, puede ser que necesitemos encapsular para enviar o compartir información.
- View: las vistas de elementos que son parte de la interfaz gráfica, como los mapas, cuadros de texto, etc.
- Notification Manager: administra las notificaciones.
- Package Manager: administra los paquetes y nos permite el uso de archivos en otros paquetes.
- Telephony Manager.
- Resource Manager: gestiona sonidos, imágenes.
- Location Manager: gestiona la posición geográfica.
- Sensor Manager: gestiona los sensores que tenemos en el dispositivo.
- Multimedia: gestiona lo referente a vídeo, audio e imágenes.

Aplicaciones (Capa AZUL)

La última capa de la imagen si vamos en el orden abajo -> arriba tiene como objetivo interactuar con el usuario ya que cuenta con las aplicaciones nativas del sistema operativo, por ejemplo, la parte de los contactos también cuenta con la administración de los escritorios, accesos directos, widgets, etc.

3 ANDROID STUDIO

Hagamos un breve repaso sobre la historia de las herramientas del desarrollo en Android.

Durante mucho tiempo no existió una herramienta exclusiva para el desarrollo Android como lo tiene iOS (sistema operativo del iPhone), se usaban herramientas que en base a algunos módulos descargables se adaptaban para Android; algunas de esas herramientas eran Netbeans y Eclipse. Motorola lanzó en sus inicios una herramienta también para desarrollar aplicaciones, pero no tuvo demasiada fuerza en comparación con las primeras mencionadas.

Netbeans

Netbeans era de las herramientas disponibles la que más compleja configuración presentaba, pero cuenta con una comunidad Java que la prefiere por encima de Eclipse, así que la mayoría de los desarrolladores Java con experiencia optaron por este, aunque con el paso de los años Eclipse fue pisando terreno hasta lograr ser la herramienta preferida de los desarrolladores Android.

Eclipse

Eclipse es uno de los IDE más sofisticados que vamos a encontrar en el mundo del desarrollo, además de su gran tamaño, complementos, configuraciones, cuenta con una de las comunidades del desarrollo ya que se puede programar en casi cualquier lenguaje si se configura este IDE de la forma correcta, en los últimos 3 años esta fue la herramienta preferida por los desarrolladores Android hasta la aparición del IDE oficial para Android.

Android Studio

Este entorno de desarrollo fue presentado en el año 2013 en el Google I/O y pasó a tomar la batuta como la herramienta, este IDE fue desarrollado por Google, compañía propietaria de Android, así que es el IDE oficial de desarrollo para aplicaciones Android.

Está basado en IntelliJ por lo que cuenta con diferencias notables en comparación con Eclipse, esto no hace ni más difícil ni más sencillo el desarrollo de una aplicación, en realidad cambian algunas cosas como la compilación, uso de librerías o paquetes externos, pero el código de una aplicación en su mayoría funciona de la misma forma sin importar el IDE que se esté utilizando.

¿Qué se necesitará para desarrollar en Android?

Vamos a responder primero la duda: ¿Qué necesitamos para desarrollar en Android? Para poder continuar con el libro, debemos saber que necesitamos 3 partes fundamentales:

- Java - Lenguaje de programación para programar en Android
- Entorno de desarrollo - Herramienta para desarrollar aplicaciones
- Android – Librerías de Android

Obteniendo Java

Java es el lenguaje de programación utilizado para desarrollar una aplicación Android, así que lo primero que debemos obtener es Java, para eso vamos a la página de Oracle y buscamos en descargas, te dejo el enlace en el que puedes encontrar Java, pero esto puede cambiar, así que lo más recomendable es ir a un buscador y teclear “Java Download” o “Java JDK”.

Es importante mencionar que no necesitamos el Java habitual que usa el navegador u otra herramienta de tu computadora, lo que se necesita es el Java JDK, esto es porque JDK es el kit de desarrollo y contiene herramientas específicas para la programación. El enlace es el siguiente:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Saldrá una imagen como la siguiente:

Resumen
Descargas
Documentación
Comunidad
Tecnología
Formación

Java SE Downloads



Java Platform (JDK) 8u65 / 8u66



NetBeans with JDK 8

Java Platform, Standard Edition

Java SE 8u65 / 8u66
Java SE 8u65 includes important security fixes. Oracle strongly recommends that all Java SE 8 users upgrade to this release. Java SE 8u66 is a patch-set update, including all of 8u65 plus additional features (described in the release notes).
[Learn more](#) ➔

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Products
- Third Party Licenses
- Certified System Configurations
- Readme Files
 - JDK ReadMe
 - JRE ReadMe

JDK
DOWNLOAD

Server JRE
DOWNLOAD

JRE
DOWNLOAD

Which Java package do I need?

Podemos ver que tenemos dos opciones, la opción de NetBeans y la opción de Java, no olvidemos que vamos a utilizar el IDE oficial, por lo tanto, solo necesitamos Java y el paquete de la izquierda es el que va a funcionar sin ningún problema.

Al seleccionar la opción de Java vamos a ver una ventana como la siguiente:

[Overview](#)
[Downloads](#)
[Documentation](#)
[Community](#)
[Technologies](#)
[Training](#)

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u65 Checksum
JDK 8u66 Checksum

Java SE Development Kit 8u65

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	jdk-8u65-linux-arm32-vfp-hflt.tar.gz
Linux ARM v8 Hard Float ABI	74.66 MB	jdk-8u65-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.67 MB	jdk-8u65-linux-i586.rpm
Linux x86	174.84 MB	jdk-8u65-linux-i586.tar.gz
Linux x64	152.69 MB	jdk-8u65-linux-x64.rpm
Linux x64	172.86 MB	jdk-8u65-linux-x64.tar.gz
Mac OS X x64	227.14 MB	jdk-8u65-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.71 MB	jdk-8u65-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.01 MB	jdk-8u65-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.22 MB	jdk-8u65-solaris-x64.tar.Z
Solaris x64	96.74 MB	jdk-8u65-solaris-x64.tar.gz
Windows x86	181.24 MB	jdk-8u65-windows-i586.exe
Windows x64	186.57 MB	jdk-8u65-windows-x64.exe

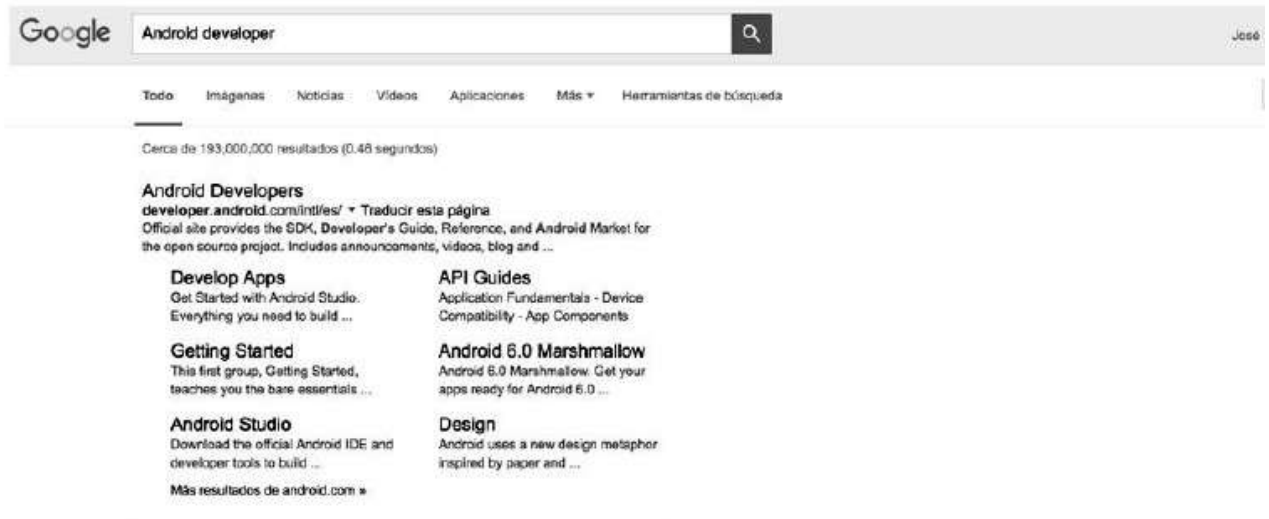
Antes de descargar una opción debemos aceptar la licencia de uso, por otro lado, debemos de seleccionar una descarga dependiendo del equipo y del procesador, por ejemplo, si tu computadora es de 64 bits y tiene Windows, tendrías que descargar el último paquete de la lista que aparece en la imagen, así que con cuidado ya que no siempre será lo mismo: depende de tu sistema operativo y de la versión de tu procesador, si es de 32 o 64 bits.

Android Studio

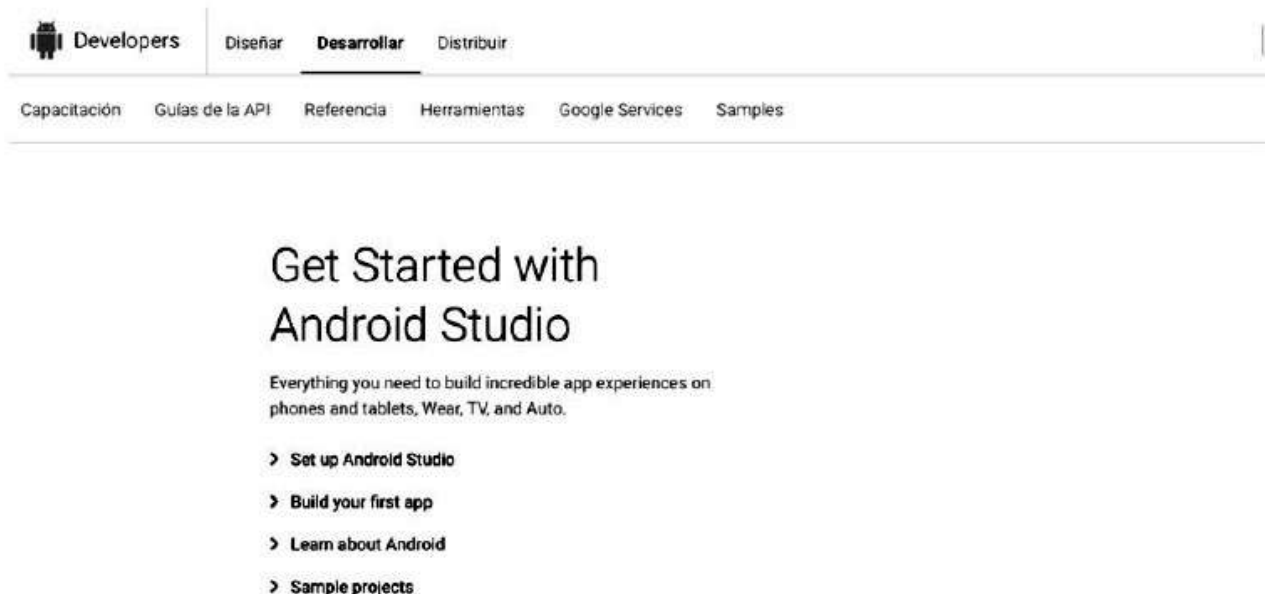
Android Studio es el IDE oficial como ya mencionamos en páginas anteriores, ahora veamos cómo obtenerlo.

Lo primero que vamos a hacer es ir al sitio oficial de desarrollo para Android:

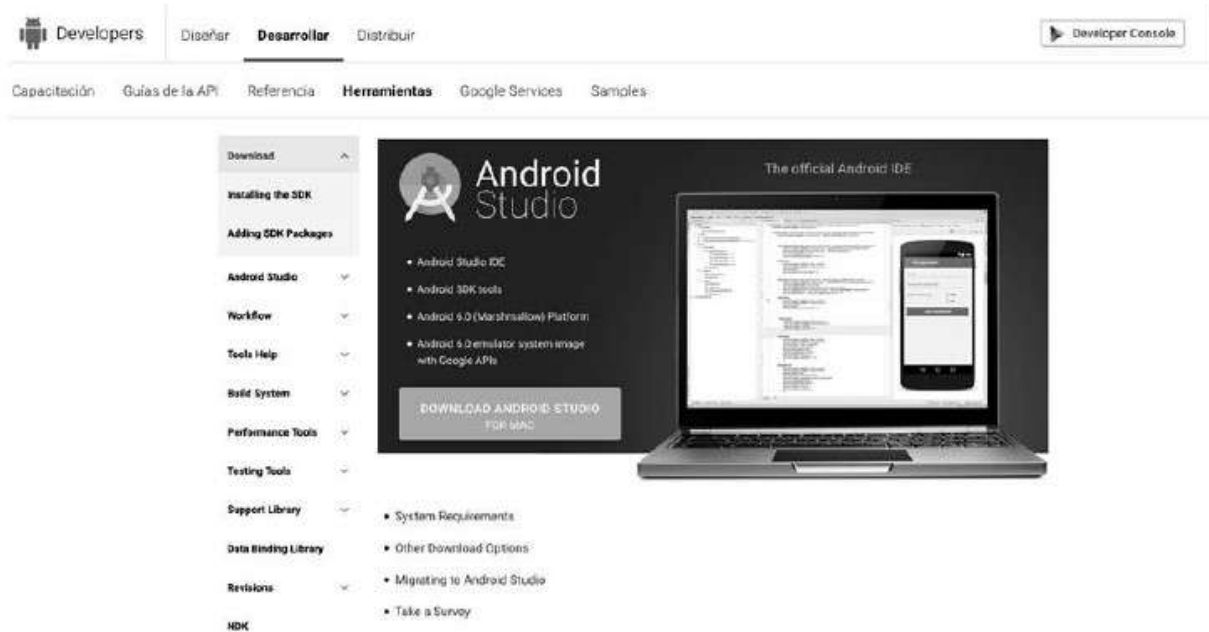
<http://developer.android.com>



Al entrar en el sitio web, seleccionamos la opción de Desarrollo para llegar a la parte de las descargas:

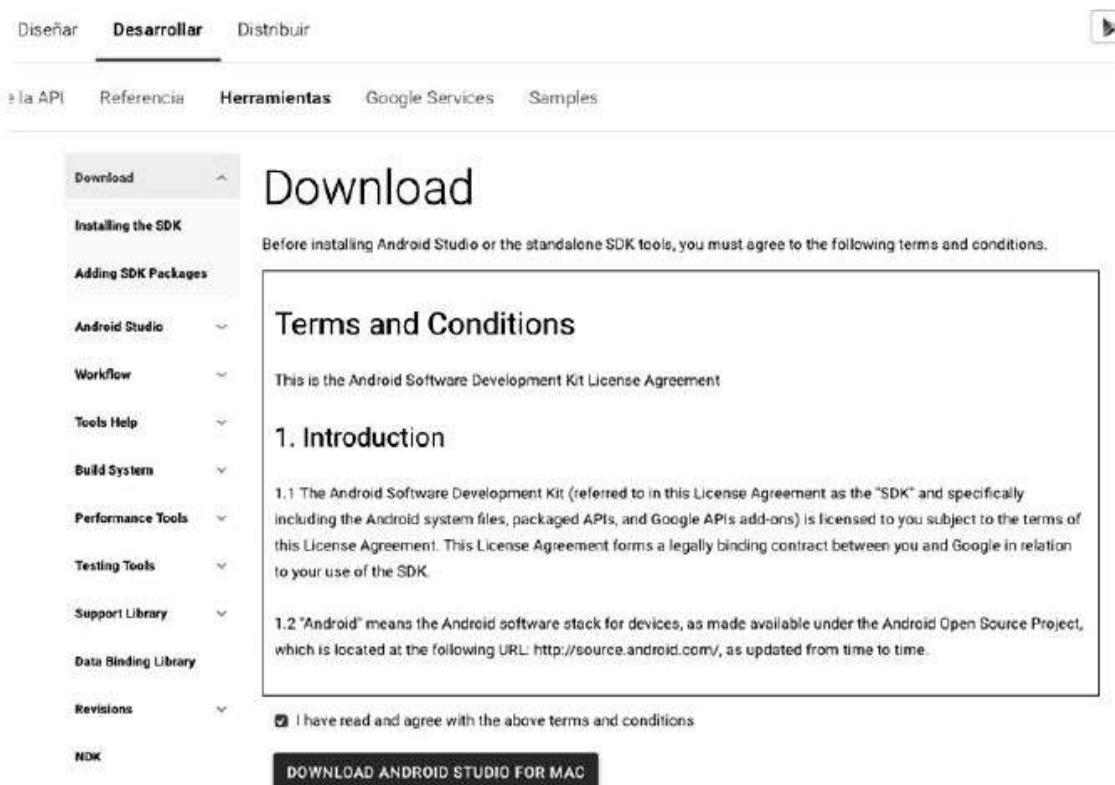


Ahora tenemos un pequeño menú con opciones en la parte central de la pantalla, vamos a la primera opción "Set up Android Studio" y vemos una ventana como la siguiente imagen:



Vemos que en la pantalla nos resalta con un botón que podemos descargar el IDE Android Studio, por defecto detecta el sistema operativo y te muestra la opción adecuada, esto quiere decir que si tienes Windows te muestra la opción para este, eso mismo hace con los otros sistemas operativos para los que existe Android Studio.

Al hacer clic en el botón veremos una pantalla como la siguiente:



Lo importante de la imagen anterior es observar que si queremos descargar Android Studio primero debemos de aceptar los términos y las condiciones para poder habilitar el botón de la descarga, ya que el botón de color azul significa que tenemos el botón habilitado y funcionando.

Al hacer clic veremos una ventana de descarga como la de cualquier programa, lo importante es que en el momento de instalar Android Studio ya tengamos Java en nuestro equipo para que el paso de instalación sea sencillo, si lo hacemos de forma correcta solo tendríamos que dar doble clic al archivo y pulsar “Siguiente” en casi todas las opciones de instalación para que se instale solo; se verá como la siguiente imagen:



Al terminar la instalación tendremos Android Studio en nuestro equipo ya casi listo para programar.

Configurar Android Studio para programar

Demos un repaso rápido a las versiones de Android, algo que debemos observar es que como desarrollador nos importa mucho más el número de la API, ya que ahí es donde reside la mayoría de las cosas que utilizamos como desarrolladores.

- Android 1.5 - API 3
- Android 1.6 - API 4
- Android API7
- Android 2.2 - API 8
- Android- API 10 API 10
- Android 3.0 - API 11
- Android API 12 API 12
- Android API 13 API 13
- Android 4.0 – API 14
- Android 4.0.3 – API 15
- Android 4.1.2 – API 16
- Android 4.2.2 – API 17
- Android 4.3 – API 18
- Android 4.4.2 – API 19
- Android 4.4W - API 20
- Android 5.0 – API 21
- Android 6.0 – API 23
- Android 7.0 – API 24

Ahora que ya recordamos las versiones de Android vamos a abrir Android Studio por primera vez.

Vamos a observar que la ventana principal se divide en 2 partes, izquierda y derecha como la siguiente imagen:



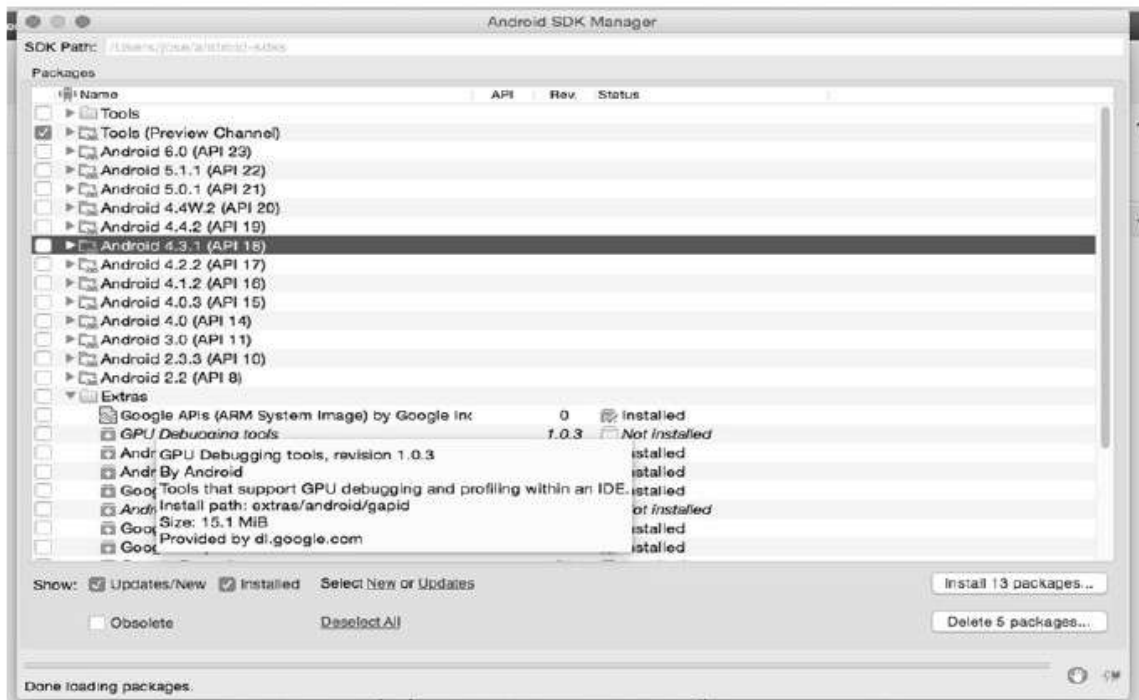
El lado izquierdo es de proyectos recientes y el lado derecho es sobre la configuración, el inicio de proyectos y otras cosas.

El siguiente paso y muy importante es obtener las versiones de Android para las que queremos programar. No es necesario descargar todas las versiones de Android existentes, pero muchos desarrolladores acostumbran a hacerlo, aunque el proceso de descargar ocupará algunos GB, es recomendable hacerlo un día que se tenga libre el uso de banda ancha ya que puede llevarte de 4 a 12 horas dependiendo de la velocidad de Internet.

Para descargar las versiones de Android seguimos las opciones siguientes:

Configuration -> SDK Manager -> Launch Standalone SDK Manager

Llegaremos a una ventana como la siguiente:



Podemos ver en la imagen anterior la API que tiene cada versión de Android y además qué paquetes ya tenemos descargados, sin descargar o una actualización pendiente.

Como mencioné no es necesario descargar todas, aunque suelen hacerlo mucho los desarrolladores novatos en Android.

Las versiones que recomiendo descargar en la fecha que se escribe este libro son las siguientes:

- API 10
- API 14-15
- API 21
- API 22
- API 23
- API 24

También el directorio de extras y el de tools.

Seguramente con estas versiones se cubrirá sin problemas lo que veamos a lo largo del libro.

Emulador

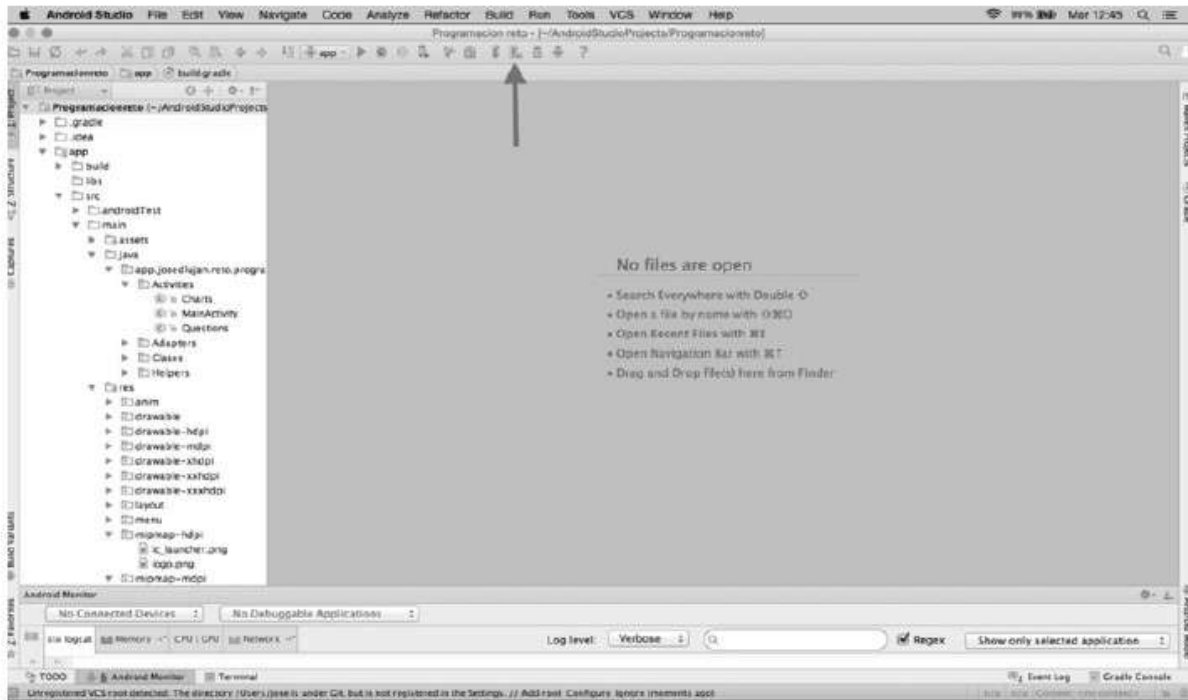
Uno de los beneficios que podemos encontrar al descargar las herramientas oficiales de desarrollo, es el poder probar nuestra aplicación en el emulador. El emulador nos ayuda a no tener que comprar la gran cantidad de dispositivos en los queremos probar nuestra aplicación, sabemos de la gran fragmentación en Android, pero esto no es un obstáculo en el momento de realizar pruebas, por eso contamos con un emulador de forma nativa en la herramienta Android Studio y no tiene ningún costo para el desarrollador.

Lo complejo del emulador es que debemos de tener una máquina preparada para su ejecución, no debemos de olvidar que un emulador significa que vamos a emular con software el hardware, así que esto es una alta demanda de recursos para nuestros equipos de desarrollo. Por recomendación Android Studio te recomienda tener 4 GB de RAM, yo te diría que eso depende, pero te doy unas recomendaciones:

- En Windows que no sea versión 10 te recomendaría tener 6-8 GB.
- Si tienes Windows 10, puedes darte el lujo de tener menos de esa cantidad y verás un funcionamiento similar.
- En el caso de Mac puedes tener 4 GB y si comparas el tiempo de arranque del emulador, notarás que es más rápido en comparación con Windows.
- En Mac el problema es al instalar la última versión del sistema operativo actual. El Capitán hace que el funcionamiento de Android Studio y el emulador en algunos casos se vuelvan más lentos.

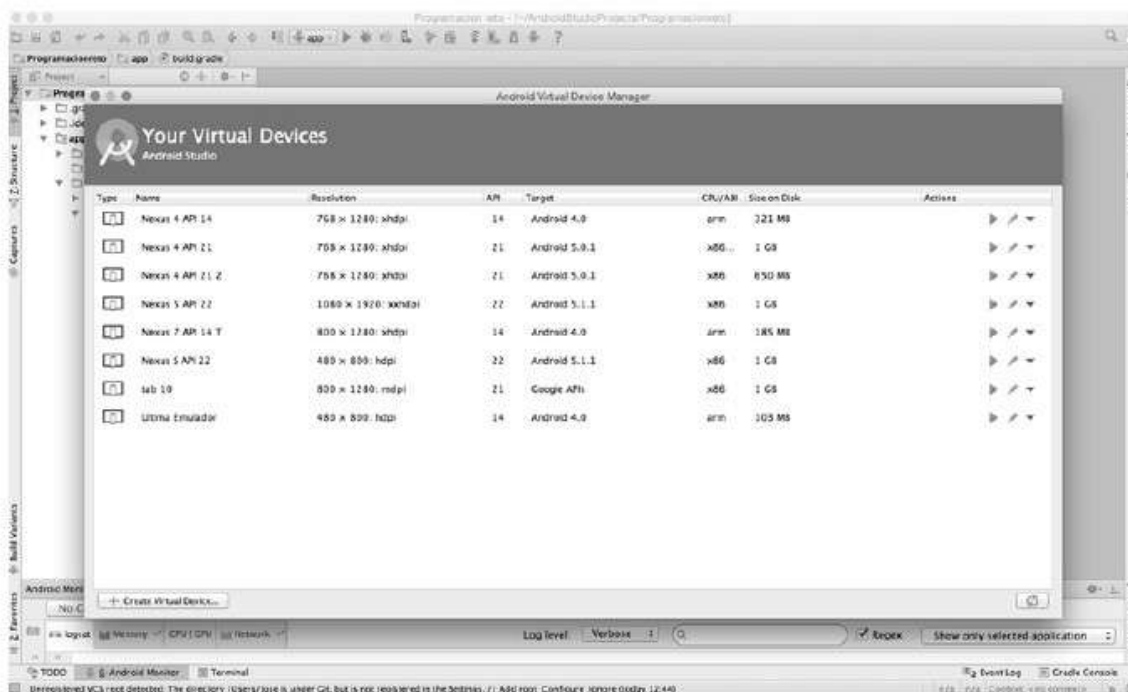
Las recomendaciones anteriores en los puntos son en relación con mi experiencia propia y además recordemos que los sistemas operativos sufren actualizaciones, también Android Studio las tiene, así que en cualquier momento se pueden mejorar o arreglar los problemas que se tienen en la actualidad, como se mencionó en otros temas del libro la recomendación principal es: tener siempre actualizado Android Studio para lograr el funcionamiento óptimo de esta herramienta.

Ahora veamos cuál es el botón para abrir el emulador en Android Studio.



Podemos ver que los iconos tienen el logo de Android, el androide verde llamado “Andy” y un dispositivo con forma de teléfono y color morado, ese es el icono que abre el administrador de los emuladores en Android Studio.

Al hacer clic veremos una ventana como en la siguiente imagen:

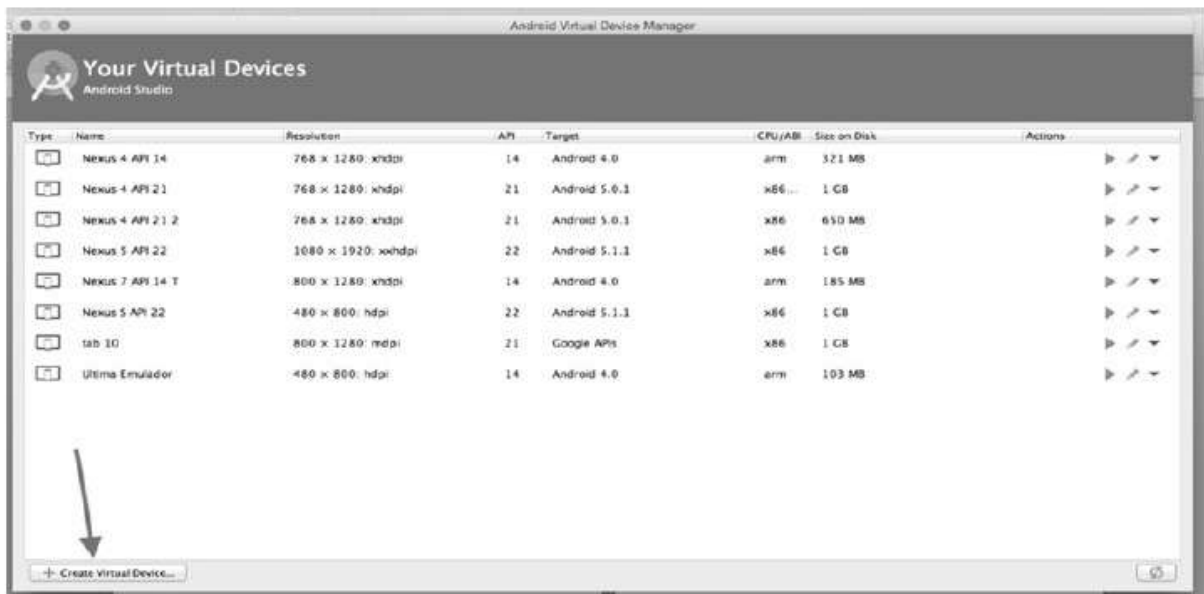


A la ventana se le conoce como AVD. El AVD significa “Android Virtual Device Manager” o en español, Administrador de dispositivos virtuales.

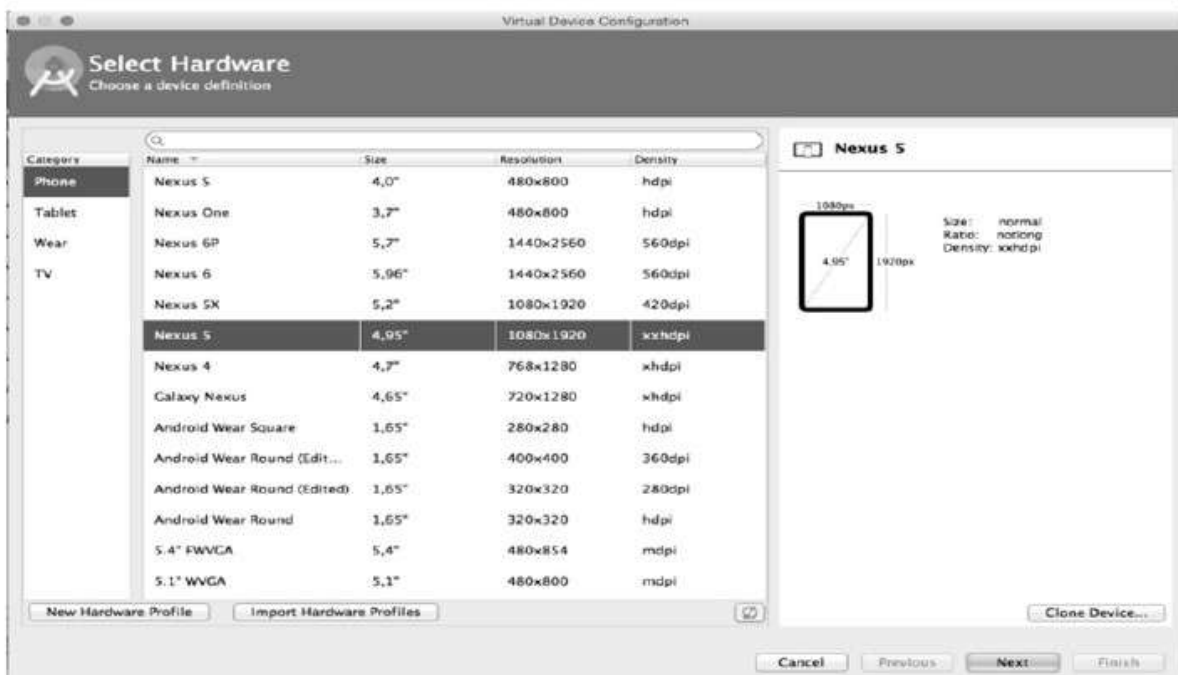
Esta ventana es la que nos permite administrar los dispositivos que ya tenemos preconfigurados, si es la primera vez que abres Android Studio no vas a encontrar dispositivos configurados y se va a ver totalmente blanco.

Configurar un emulador

Vamos a hacer clic en el botón “Create Virtual Device” que señalamos en la imagen.



Al hacer clic, vemos la siguiente ventana:



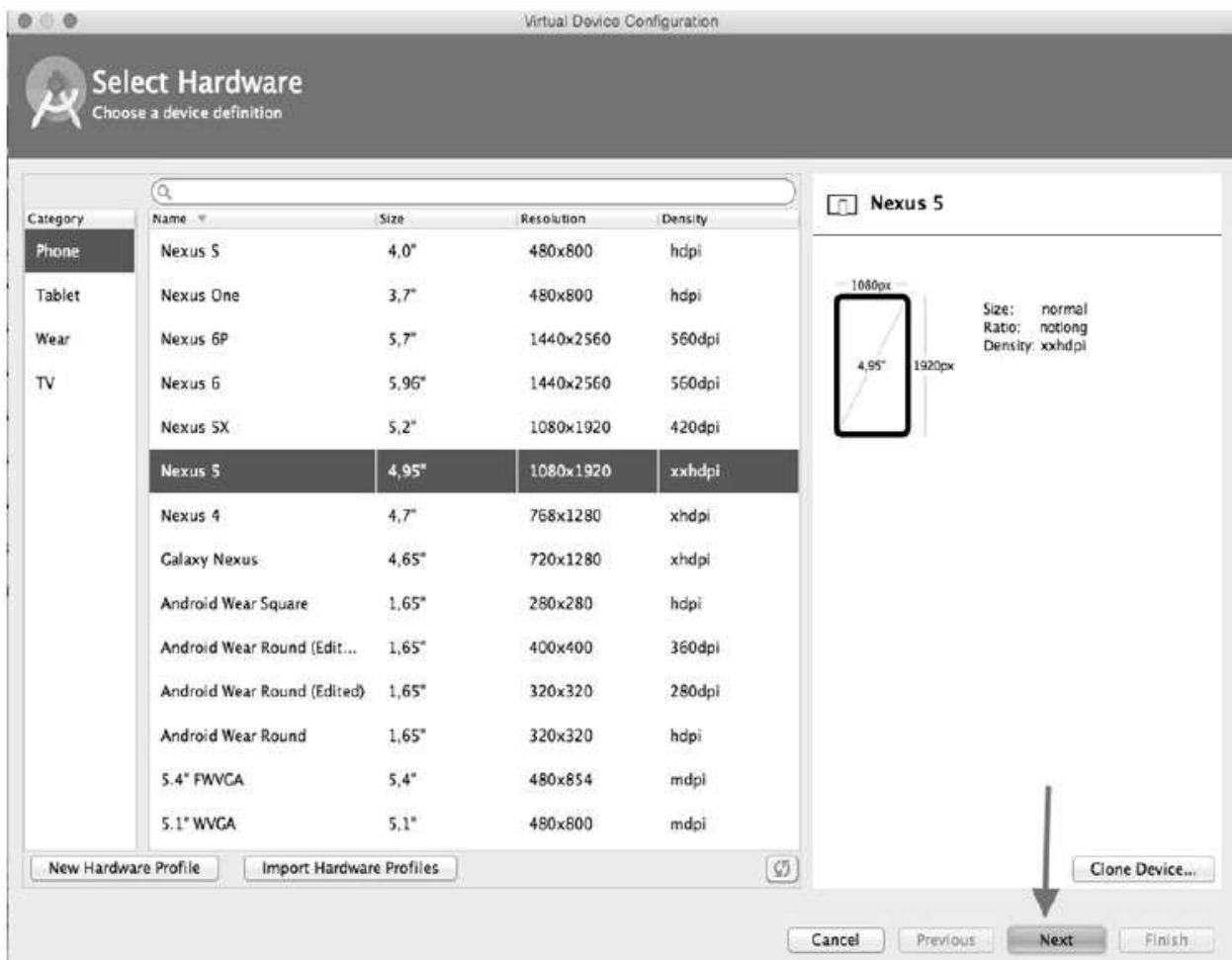
Podemos ver un menú principal que nos permite crear 4 tipos de dispositivos:

- Teléfono
- Tablet
- Reloj
- TV

Android Studio te va a permitir crear un emulador para cualquiera de los dispositivos mencionados anteriormente, pero también te da por defecto algunas configuraciones, por ejemplo:

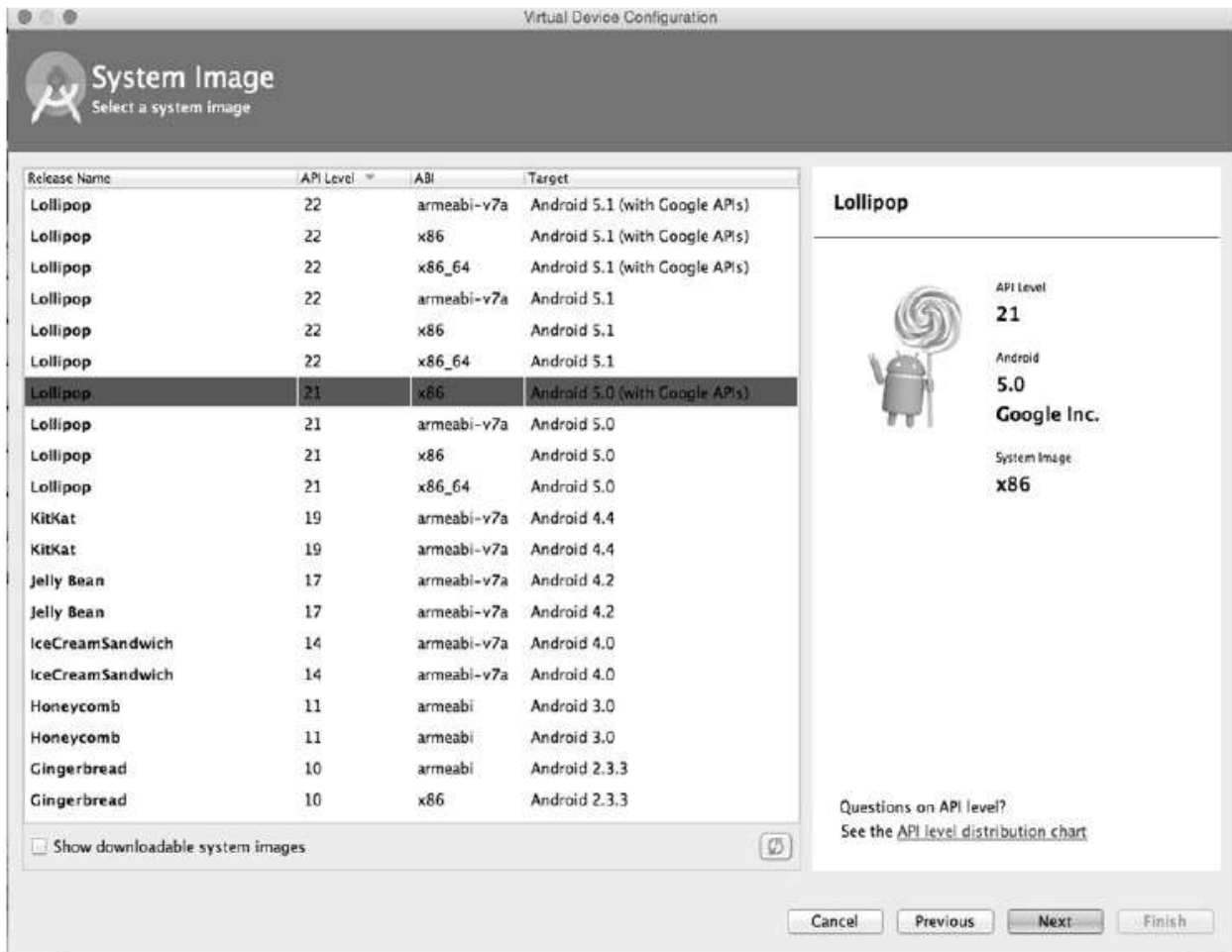
Si seleccionas un teléfono te muestra la familia Nexus como preconfigurados: Nexus S, Nexus One, Nexus 6p, Nexus 6, Nexus 5x, Nexus 5, Nexus 4.

También te coloca algunas preconfiguraciones por tamaño de pantalla y/o por densidad, en nuestro caso vamos a crear de ejemplo un Nexus 5.



Vamos a seleccionar Nexus 5 y damos clic en el botón Siguiente.

Al hacer clic vemos una ventana en donde tenemos que seleccionar la versión de Android que le vamos a colocar al dispositivo que estamos creando, en nuestro caso seleccionamos la versión 5.0 como la siguiente imagen y damos clic en Siguiente:



Ahora vamos a configurar las siguientes opciones:

AVD Name: aquí colocamos el nombre con el que vamos a identificar al dispositivo que estamos creando, se recomienda colocar el número de la API y el modelo del teléfono para identificar de forma rápida la configuración.

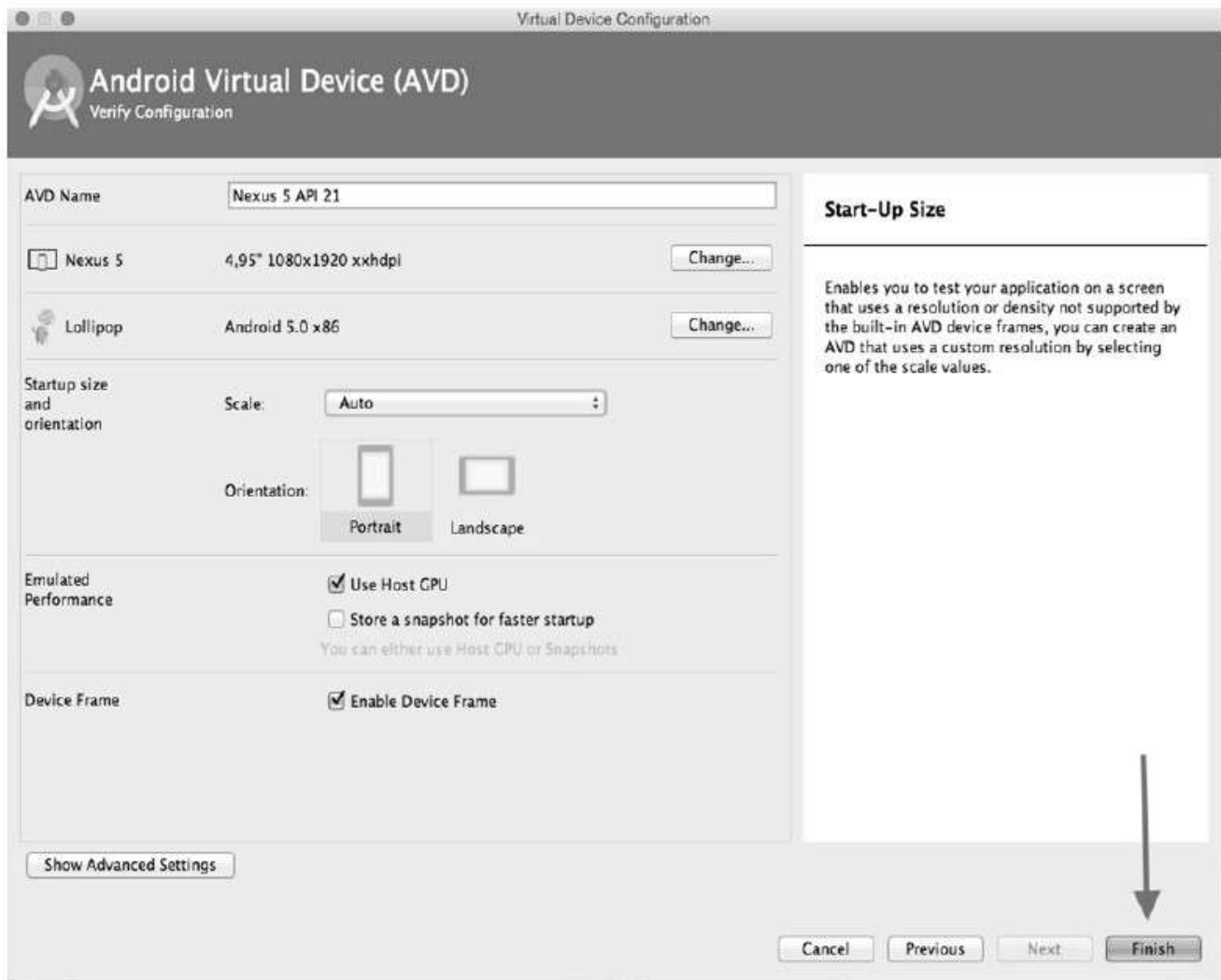
Scale: en esta opción podemos cambiar la escala para ver el dispositivo.

Orientation: por defecto en Android tenemos 2 orientaciones: Portrait (Vertical) y Landscape (Horizontal). Podemos seleccionar en este punto la orientación con la que queremos iniciar el dispositivo, aunque la podemos cambiar sin ningún problema.

Snapshot. El emulador va más rápido, cuando cierras el emulador la siguiente ocasión se ejecutará más rápido porque tiene el último estado antes de cerrar.

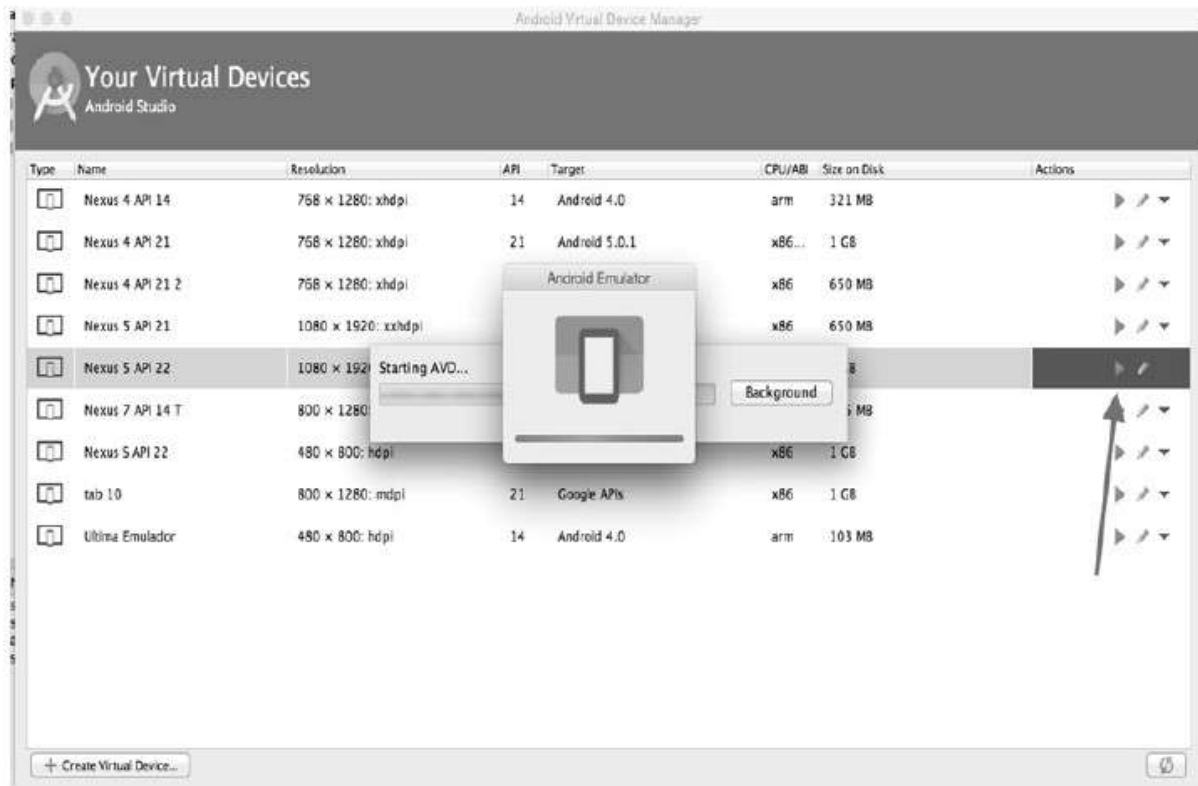
Host GPU. El emulador utilizará el GPU del PC, sirve para que el emulador sea mucho más fluido ya que el procesamiento de gráficos lo realiza el PC.

Ahora le damos clic en el botón Finalizar para acabar con la configuración.

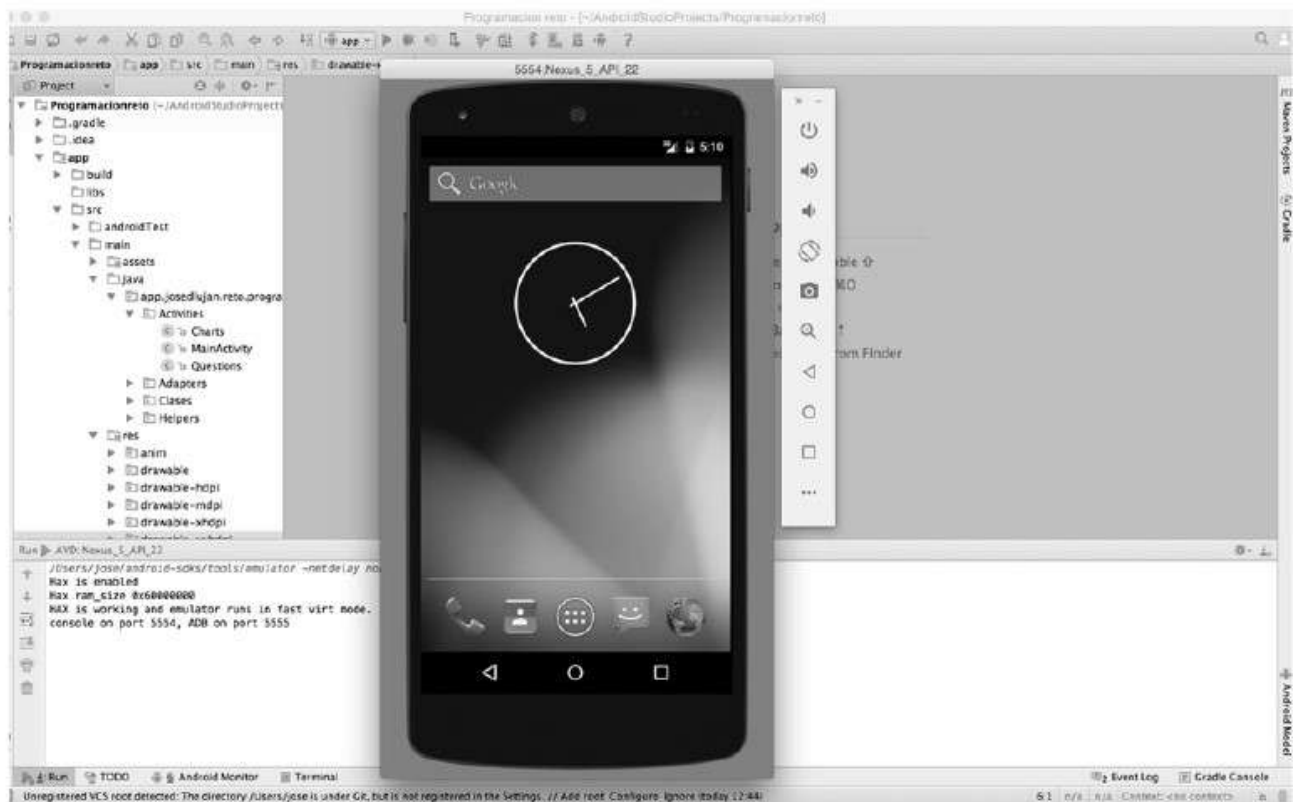


Con eso ya terminamos finalmente de configurar nuestro dispositivo.

Para arrancar el emulador abrimos el AVD manager y seleccionamos el botón Play que tiene la configuración que deseamos del lado derecho.



Después de esperar un par de segundos o minutos dependiendo de la potencia de tu computadora vamos a ver el emulador funcionando.



Finalmente tenemos un emulador para probar nuestras aplicaciones.

La versión actual con la que contamos es la versión Android Studio 2.2.

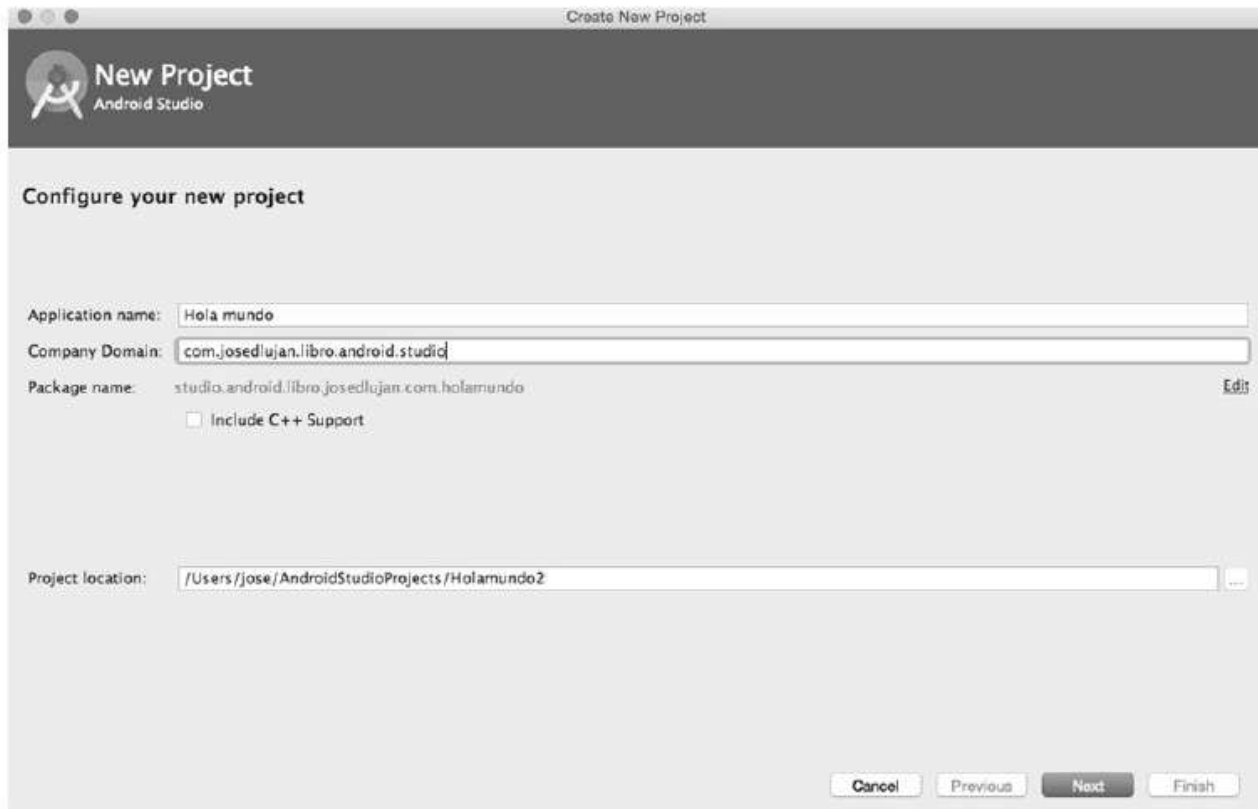
4 CREAR UNA APLICACIÓN

Crear una aplicación

Lo primero que tendremos que hacer es abrir Android Studio:



Vamos a seleccionar la primera opción "Start a new Android Studio project", ahora veremos una ventana como la siguiente.



Describamos las opciones:

Application Name

Colocaremos el nombre de la aplicación, ten cuidado al seleccionar un nombre ya que es el que aparecerá en la playstore en caso de publicarla, como es un “Hola Mundo” pondremos eso como nombre.

Company Domain

Colocaremos el dominio de la compañía, normalmente inicia con “com” en este caso lo llamaremos: com.josedlujan.libro.android.studio

El nombre no debe llevar espacios y debemos de tratar que sea único, este company domain será parte de la url de nuestra aplicación en la tienda, por eso, debemos tratar de que sea único. Todo debe ir en minúsculas.

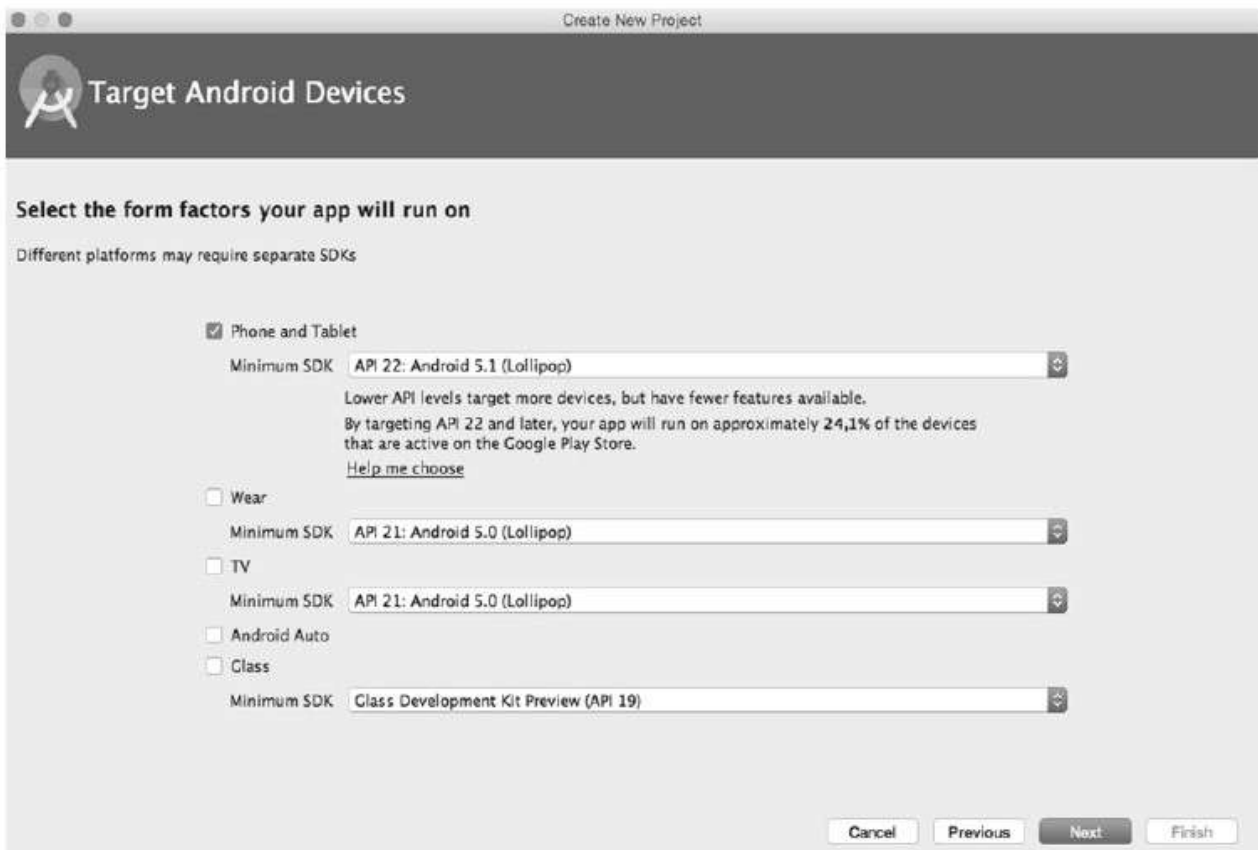
Package Name

El nombre del paquete también debe ser único. Podemos dejar el valor que se coloca por defecto que es lo más recomendable, pero si por alguna extraña razón se tiene que cambiar tenemos la opción que dice que lo podemos editar.

Project Location

Este apartado nos permite colocar la ubicación física de los directorios y archivos que tendrá la aplicación, normalmente se deja lo que se coloca por defecto, así que te daría esa recomendación, pero en algún caso extraordinario sin problemas puedes modificar la ubicación solamente asegúrate de recordar la ubicación del proyecto.

Finalmente damos clic en Next y vemos la siguiente ventana:



Tenemos las siguientes opciones:

Phone and Tables, Minimum SDK

En el caso de que nuestra aplicación vaya a ser para un teléfono o una tablet tenemos que marcar esta opción, además de marcarla debemos de seleccionar dentro de las opciones de Minimum SDK. Tendremos que seleccionar la versión mínima de Android que va a soportar nuestra aplicación, en otras palabras, estamos diciendo cuál es la última versión de Android que va a poder ejecutar nuestra aplicación, lo ideal es dejar que Android te diga qué versión es. Eso se hace dando clic a la opción de debajo que dice “Help me choose”, normalmente ahí vemos que las versiones convenientes son las que abarquen casi el 90% del mercado, esto cambia de manera rápida, así que siempre apóyate en la ayuda de Android Studio para tomar esta decisión.

Wear

En este apartado sucede lo mismo que en el primero, pero es para el caso de querer hacer una aplicación con Android Wear.

TV

Este apartado es igual que todos los demás, lo único es que debemos de seleccionar la opción que veamos mejor preparada para Android Studio.

Glass

En el caso de querer desarrollar una aplicación para Google Glass, la herramienta que se tendrá que usar es Android Studio y desde aquí tendrías que seleccionar la versión mínima del SDK que va a ser compatible con tu proyecto.

Ya que se seleccionó el tipo de aplicación y número de la versión con la que se va a trabajar damos clic en el botón “Next”.

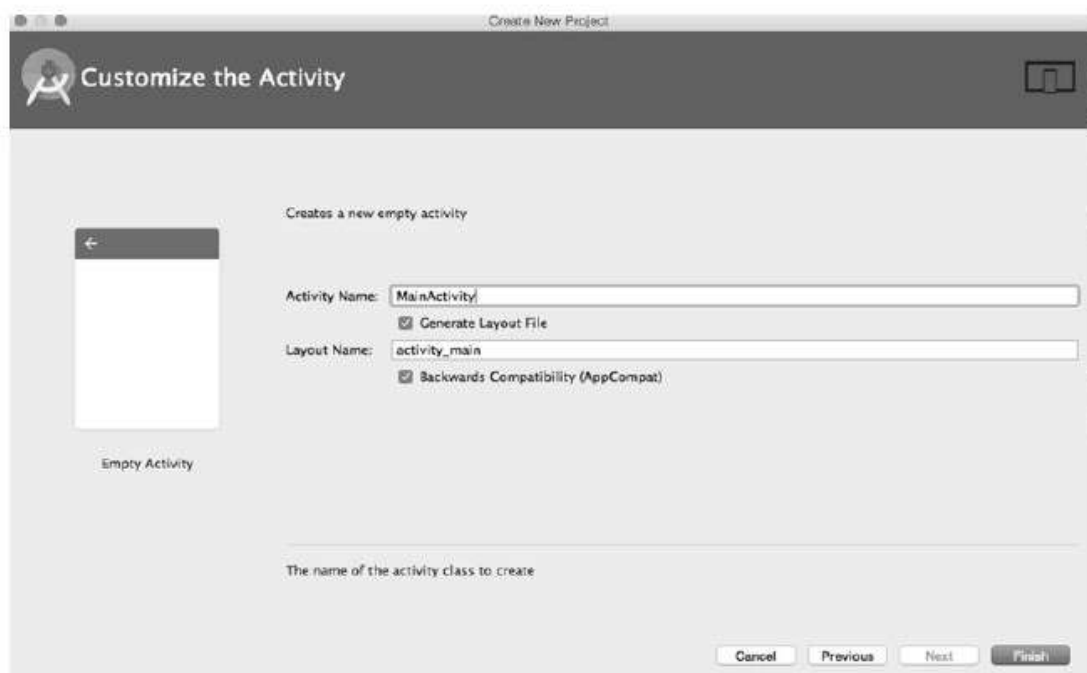
Ahora se abre una ventana como la siguiente:



Lo que observamos es que ahora podemos seleccionar la primera ventana de nuestra aplicación, podemos decir que en Android las ventanas no existen, el nombre que se utiliza para describir un elemento como el que vemos en pantalla es el de “Actividad” o Activity en inglés.

En nuestro caso vamos a utilizar la opción “Empty Activity” en este caso nos va a crear una actividad vacía, es decir, que vendrá con la cantidad mínima de código posible.

Al seleccionarla hacemos clic en Next y veremos la siguiente ventana:



Podemos ver una ventana que nos permite crear nuestra primera actividad, lo normal es colocarle un nombre a la activity.

Activity Name

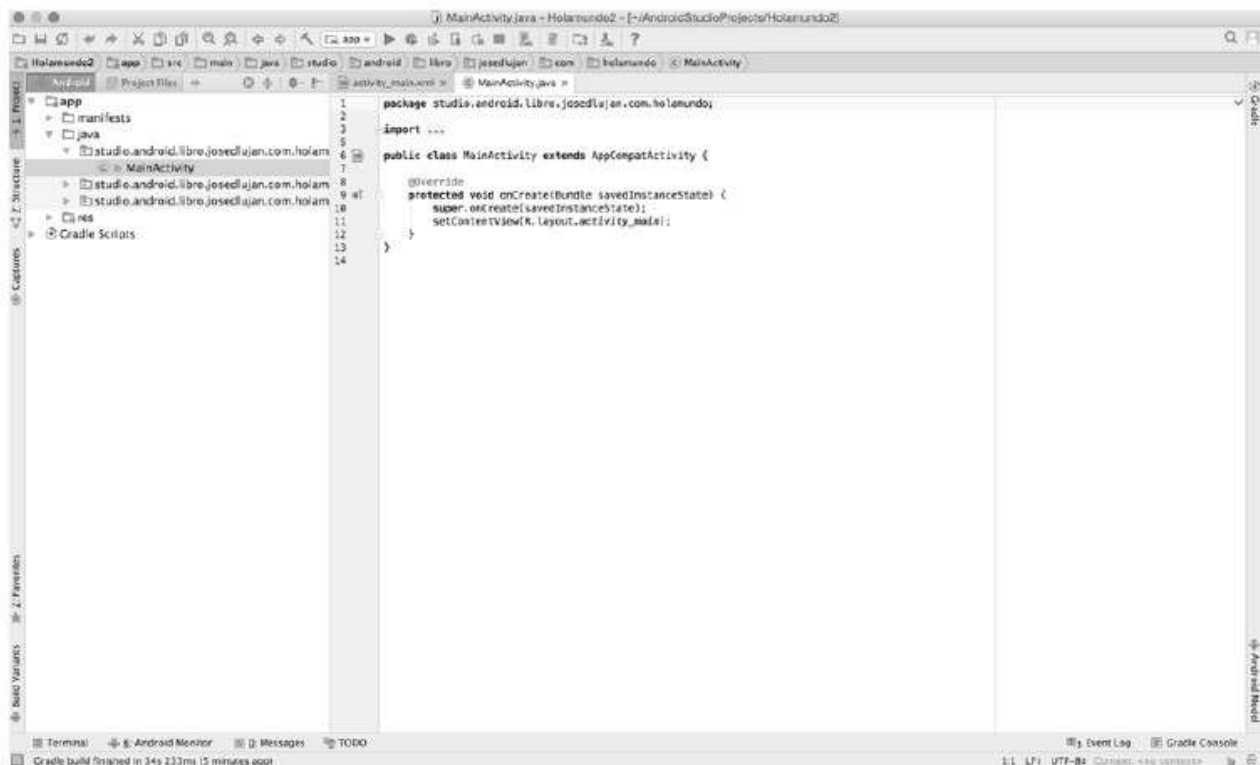
Es el nombre de la actividad y quiere decir que es el nombre de la clase que controla la primera pantalla, por convención la primera “activity” de una aplicación se llama MainActivity.

Layout Name

Este es el nombre de la maquetaación o del archivo que contiene los elementos gráficos de la pantalla con la representación gráfica de la actividad, son todos los elementos gráficos que se ven en nuestra aplicación.

Ahora hacemos clic y el proyecto se empieza a crear.

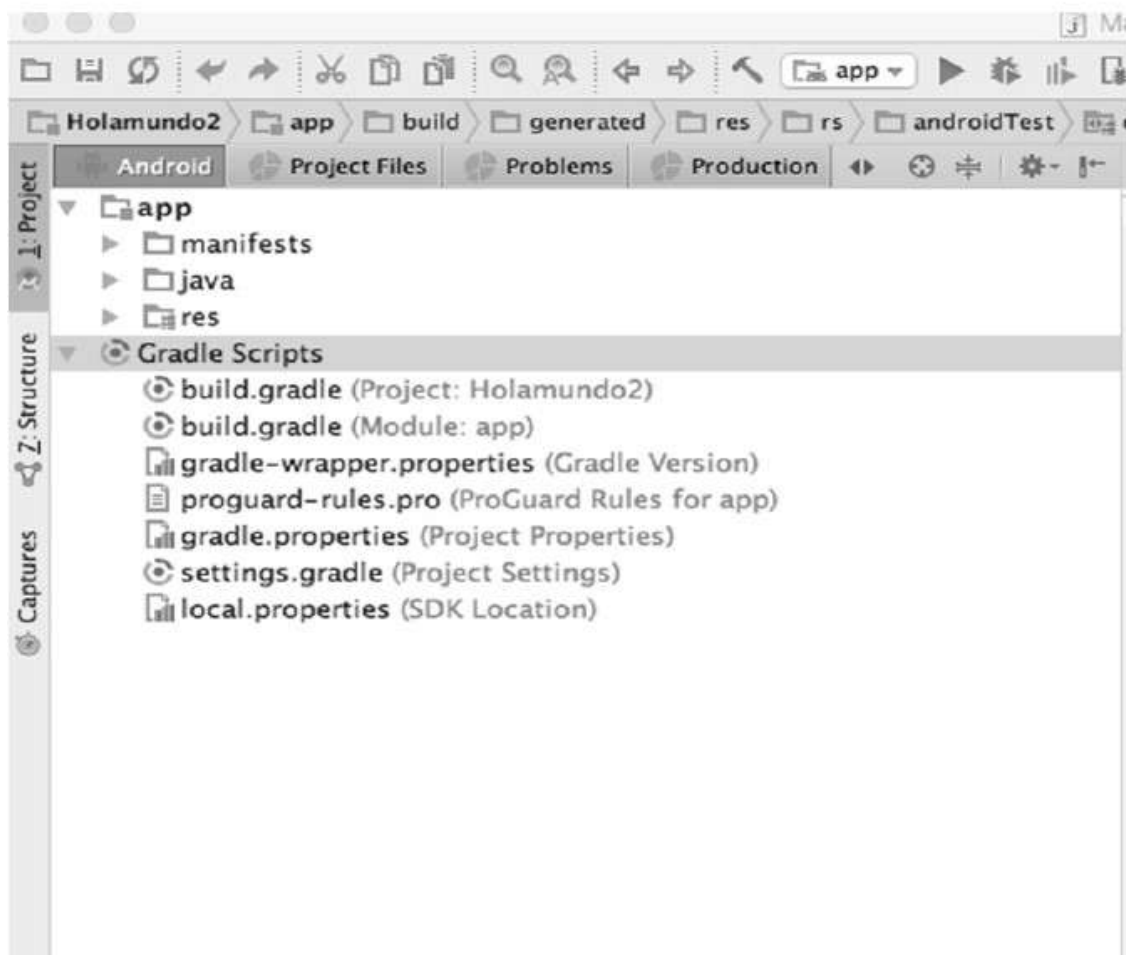
Veremos la siguiente ventana:



En realidad, para el “Hola Mundo” no tenemos que hacer nada en Android Studio, por defecto se crea esa instrucción y únicamente tuvimos que crear un proyecto vacío, con este terminamos el capítulo de la construcción de un “Hola Mundo”.

5 ESTRUCTURA DE UN PROYECTO

Tomando como referencia el final del capítulo anterior en el que se construyó el “Hola Mundo” vamos a tomar esa estructura de proyecto y explicarla; la estructura se debería de ver como la siguiente imagen:



Perspectivas

En la imagen anterior podemos ver que en la parte superior en donde está la estructura del proyecto aparecen las siguientes opciones:

- Android
- Project Files
- Problems
- Production

Tenemos más opciones, pero estas son las que se ven en la imagen, y se las conoce como perspectivas, esto quiere decir que un proyecto o una aplicación se puede ver de diferentes formas dependiendo de las necesidades y de lo que vayamos a modificar.

La perspectiva por defecto es **Android** como lo tenemos en la imagen, pero otra perspectiva muy usada es la de **“Project Files”**, ambas son muy utilizadas en el desarrollo de cualquier aplicación por muy básica o avanzada que sea.

Estructura de un proyecto

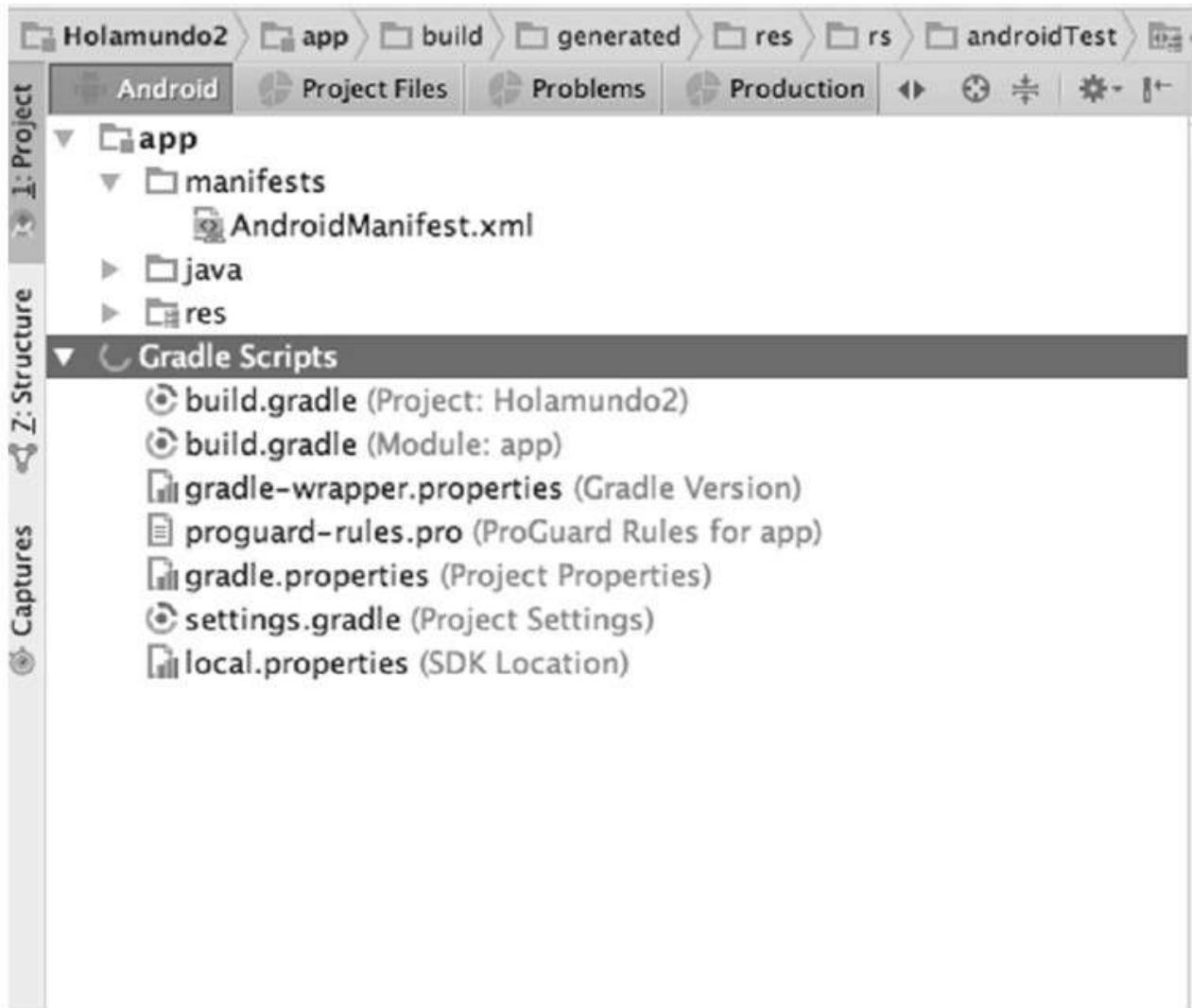
Vamos a ver la estructura de un proyecto a partir de la estructura Android, partiendo de la imagen en la que se pudo ver la estructura, podemos decir que tenemos como directorio principal el directorio **“app”**.

Dentro del directorio **“app”** tenemos una serie de directorios que podemos considerar como los principales dentro de una aplicación Android que son:

- Manifests
- Java
- Res
- Gradle Scripts

Directorio manifests

Dentro del directorio manifest podemos observar los siguientes archivos:



El archivo *AndroidManifest.xml* es el que vamos a encontrar dentro de este directorio. Este archivo es uno de los más importantes dentro de una aplicación Android, en él encontraremos instrucciones que son indicaciones de configuración para la aplicación que estamos haciendo.

Si abrimos el archivo Manifest, veremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="studio.android.libro.josedlujan.com.holamundo">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```

        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>

```

El archivo se encuentra en formato XML y como cualquier archivo XML nos indica en su primera línea el formato, la versión y la codificación:

```
<?xml versión="1.0" encoding="utf-8"?>
```

La siguiente línea hace referencia al esquema y al nombre del paquete, este lo definimos al inicio cuando creamos el proyecto, estas líneas siempre estarán por defecto creadas por Android Studio, pero nunca está demás saber qué dice:

```

<manifest
xmlns:android="http://schemas.android.com/apk/res/Android"
    package="studio.android.libro.josedlujan.com.holamundo">

```

La etiqueta aplicación contiene la configuración de la misma en general, es decir, que se asignará a toda la aplicación, la línea siguiente es un valor booleano en el que se da o no se da permiso para que se pueda hacer un respaldo de nuestra aplicación en un dispositivo Android con alguna aplicación de respaldo, si se coloca True estamos permitiendo hacer el respaldo, en caso de poner False lo estaríamos prohibiendo.

```

<application
    android:allowBackup="true"

```

La siguiente línea es en la que indicamos el icono de nuestra aplicación, más adelante veremos cómo cambiar el icono de nuestra aplicación, todas las aplicaciones Android toman el icono por defecto que se llama “ic_launcher”.

```
android:icon="@mipmap/ic_launcher"
```

En la siguiente línea se indica el nombre de la aplicación, este nombre se encuentra en un archivo de cadenas de texto y lo único que se hace es referenciar la cadena dentro de ese archivo de cadenas, por defecto se llama “app_name” pero se puede crear una con un nombre diferente y simplemente cambiarlo.

```
android:label="@string/app_name"
```

La línea siguiente es de las instrucciones más nuevas que se han colocado por defecto en el Manifest, esta línea indica que la escritura es de derecha a izquierda, esto se debe a que existen escrituras en las que se escribe de forma contraria, en este caso por defecto es de derecha a izquierda y así nos sirve para trabajar, en caso de que necesitemos indicar lo contrario cambiaríamos el valor de “true” por el de “false”:

```
android:supportsRtl="true"
```

Las aplicaciones siempre cuentan con un tema base para facilitar el diseño y la compatibilidad de elementos, por defecto se coloca el tema definido como Apptheme:

```
android:theme="@style/AppTheme">
```

Por defecto nuestro “Hola Mundo” cuenta con una actividad, en el Manifest siempre vamos a tener definidas todas las actividades que tengamos en nuestra aplicación, si una actividad no está definida va a haber un error y por tanto se va a cerrar.

La actividad principal en este caso sería este bloque de código:

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

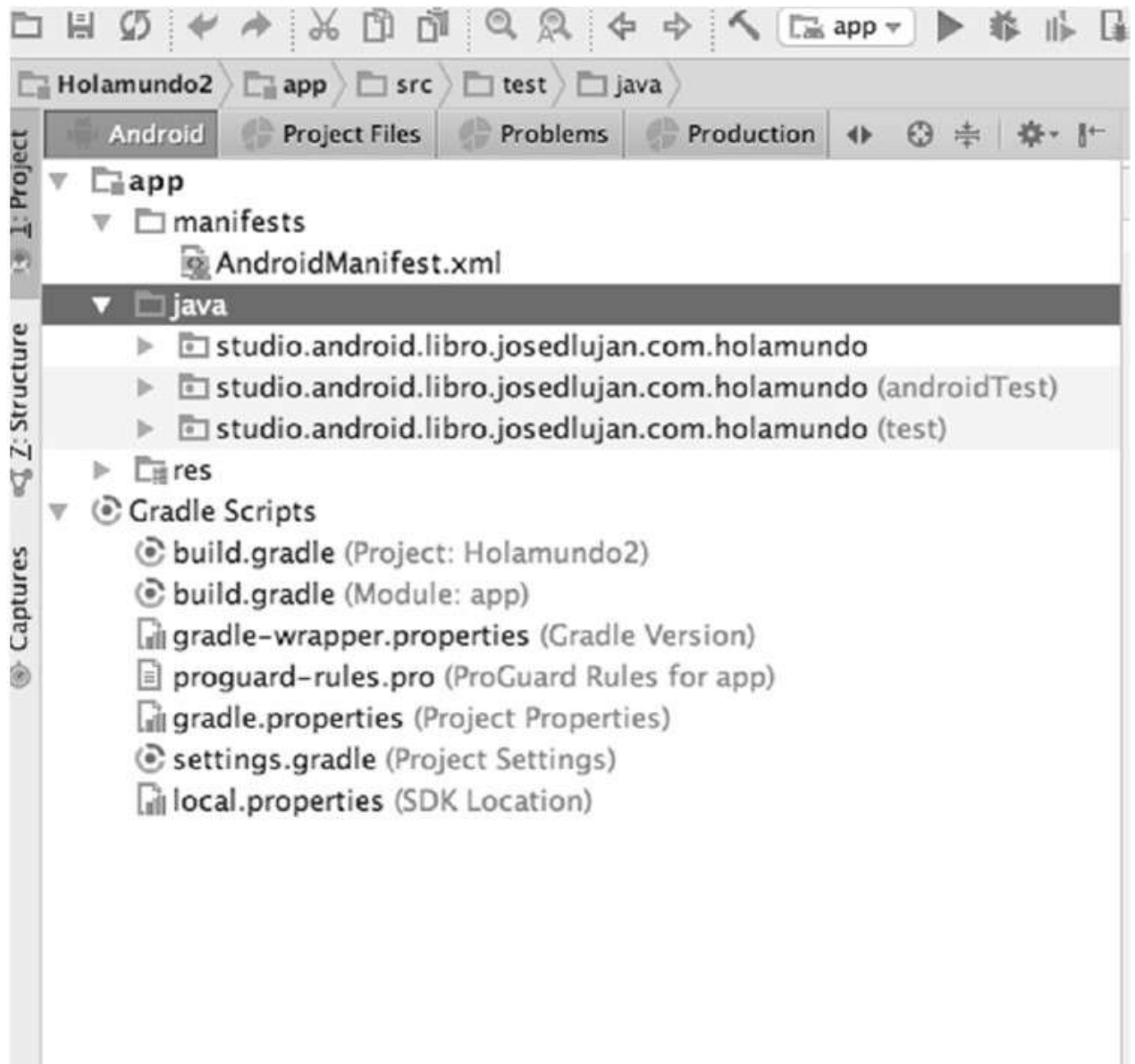
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

La primera línea indica el nombre del archivo Java que es la lógica de la aplicación, el nombre que tiene es el que le colocamos al inicio cuando creamos el proyecto que es “MainActivity”.

El siguiente bloque es un intent filter en el que definimos propiedades de la actividad, por ejemplo, el nombre de la actividad y la categoría, por defecto podemos colocar diferentes valores en la categoría, pero siempre la actividad que se va a ejecutar cuando el usuario abra la aplicación tiene que llevar el valor LAUNCHER, así es como Android sabe qué actividad tiene que mostrarse al usuario cuando este abra la aplicación.

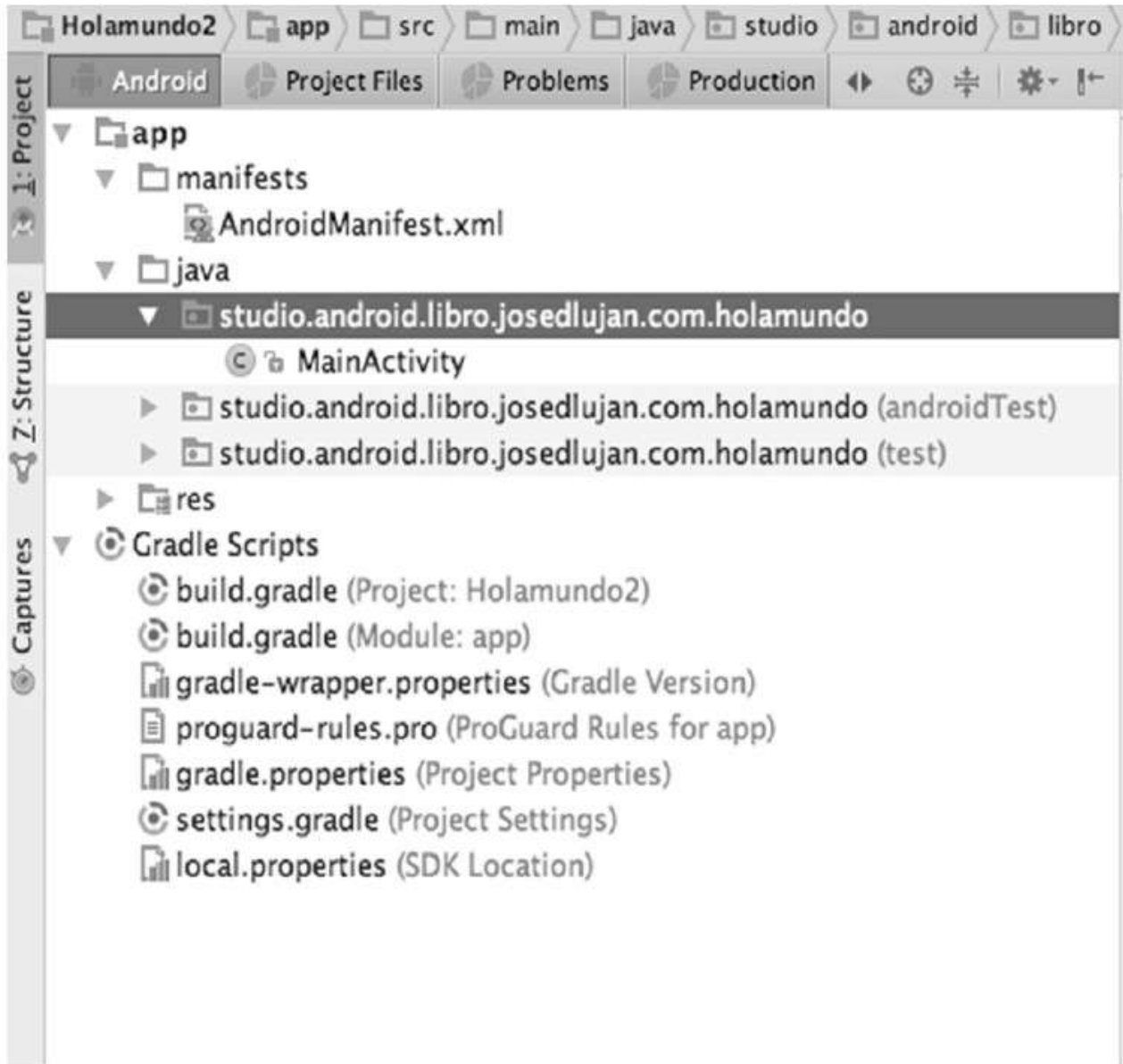
Directorio Java

Si desplegamos el directorio Java veremos lo siguiente:



El directorio java contiene 3 directorios más con el mismo nombre, la diferencia es lo que podemos ver entre paréntesis, uno no tiene nada, el otro contiene “androidText”, y el tercero contiene la palabra “test”, en realidad los dos últimos no nos interesan, el único directorio que nos interesa por ahora es el que no tiene texto en paréntesis, en este directorio vamos a guardar todos los archivos Java que tendrá nuestro proyecto, por defecto solo tenemos el directorio Main.java que es la única actividad que tenemos definida.

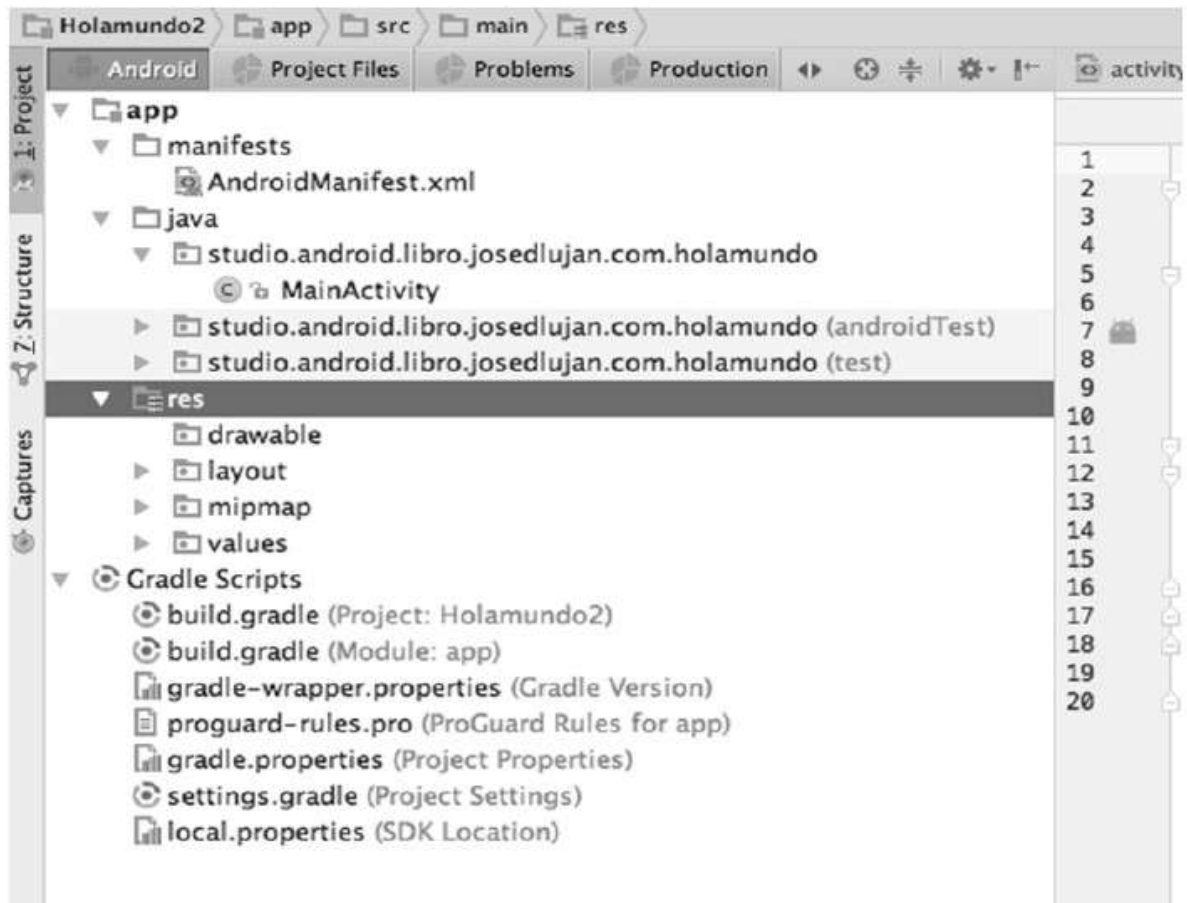
Lo podemos ver en la siguiente imagen:



Directorio res

El directorio res es el directorio con más archivos ya que es el directorio en donde se colocan todos los recursos que utiliza la aplicación, desde las cadenas de textos, imágenes, vectores y cualquier recurso que necesita la aplicación.

Si desplegamos el directorio res veremos lo siguiente:



Vamos a analizar por orden de aparición los elementos:

drawable

Este directorio contiene los archivos que son imágenes y que necesitamos tengan un identificador para después mandarlos a llamar. Por ejemplo, si colocamos un fondo de pantalla o una imagen en nuestra aplicación es seguro que se colocará en el directorio drawable, por defecto tiene este nombre, pero cuando ya se va a hacer una aplicación para distintos dispositivos tendríamos que crear diferentes directorios drawables para que la aplicación tenga todas las imágenes correspondientes para todas las densidades y tamaños de pantallas.

layout

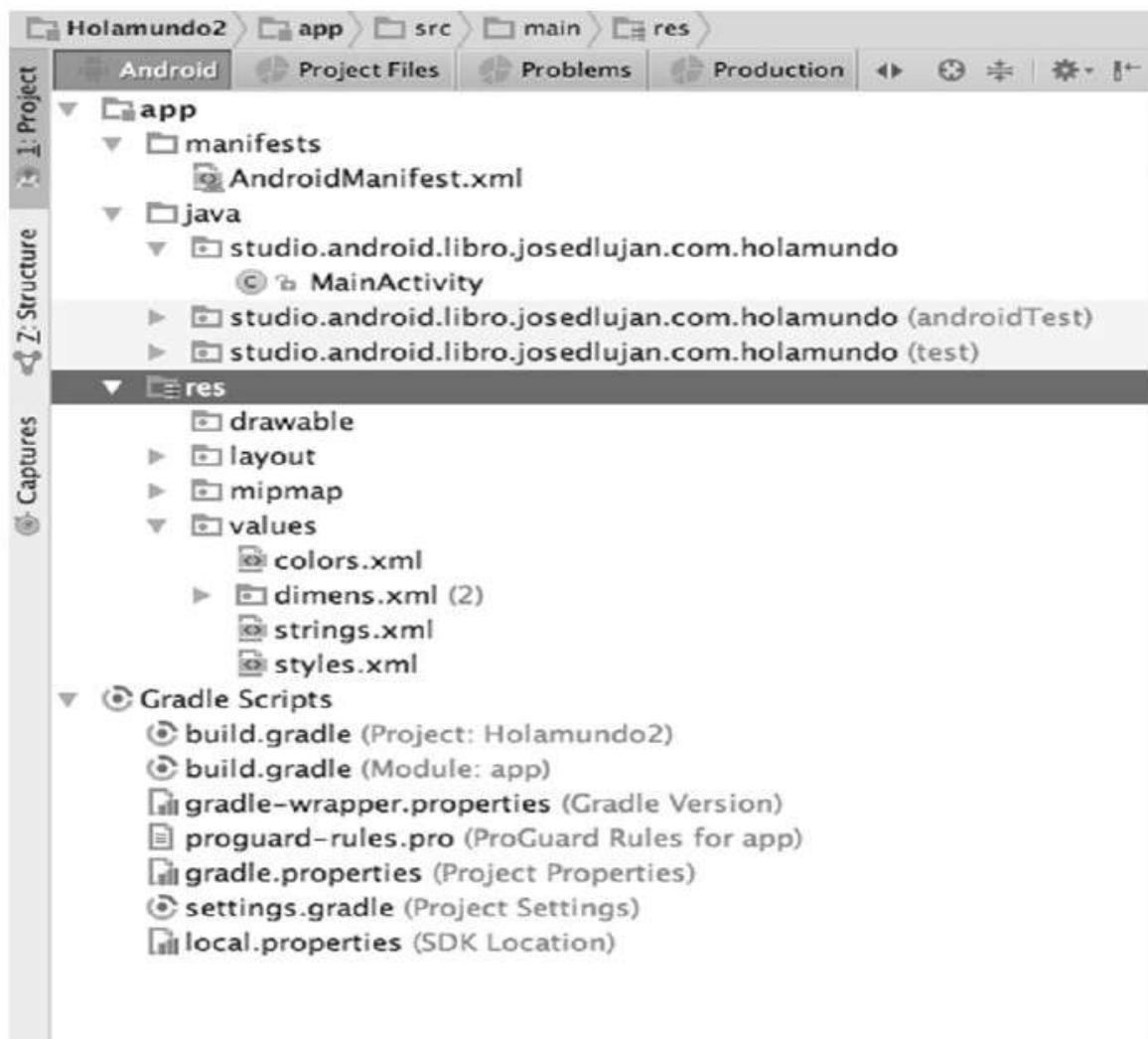
El directorio layout contiene todos los archivos que son las interfaces de las pantallas de la aplicación, por ejemplo: si tenemos actividades en nuestra aplicación, deberíamos de tener al menos 5 archivos xml que sean la interfaz de cada una de las actividades. Así que podemos decir que en este directorio se guarda toda la parte visual de nuestra aplicación.

mipmap

El directorio mipmap se utiliza para guardar el icono de la aplicación, sirve para guardar imágenes también como el directorio drawable, la diferencia es que este directorio se mantiene íntegro. El directorio que tiene el nombre drawable siempre contendrá los recursos visuales para todas las pantallas Android, pero en el momento de la instalación se detecta la instalación correcta y se borran todos los archivos que no corresponden al tamaño del dispositivo, en el caso de mipmap aunque no correspondan, se guardan ya que se utilizan para otras cosas también, por ejemplo el icono de la aplicación acostumbra utilizar una densidad más alta de la calidad de la imagen para que se vea mejor, así que en este directorio siempre encontrarás el icono de la aplicación.

values

En este directorio encontrarás diferentes valores como puedes ver en la imagen:



Tenemos valores como colores, dimensiones, cadenas de texto y estilos. Estos archivos contienen diversos valores, pero la mayoría tienen relación con el aspecto visual de la aplicación. El archivo `colors.xml` contiene colores que podemos definir en formato hexadecimal y solo mandarlos a llamar por su nombre, como sucede con HTML y CSS al hacer una página web.

El archivo `strings.xml` contiene las cadenas de texto que vamos a utilizar dentro de la aplicación, este archivo es el responsable de que podamos hacer aplicaciones multi-idioma de una manera muy sencilla.

El archivo `styles.xml` nos permite definir estilos para utilizar dentro de la aplicación, hoy en día un diseño como Material Design es muy útil para la implementación de los estilos necesarios en el desarrollo de una aplicación.

Gradle

Gradle es una tecnología que nos permite automatizar la creación de proyectos ya que podemos gestionar proyectos, paqueterías, librerías y otros elementos que se usan dentro de cualquier proyecto de software.

Una de las características más importantes de Gradle es que maneja compilación incremental, esto quiere decir que si no realizamos cambios no pasará nada, pero cuando se hace un cambio después de la última compilación, se hará una “recompilación”, así que cuando no encuentra cambios simplemente no hace la compilación.

De la estructura de nuestro proyecto aún nos falta analizar el último apartado que es el directorio de Gradle Scripts.

Gradle Scripts

El directorio de Gradle Scripts contiene scripts como su nombre indica, que es una parte de la configuración de un proyecto en Android Studio. Gradle es una herramienta de automatización que nos facilita el manejo de paquetes, librerías y dependencias.

En este caso nos importan los dos archivos que llevan por nombre `build.gradle`, pero entre paréntesis uno es `Project` y el otro `Module`.

build.gradle (Project)

El archivo build.gradle (Project) tiene el siguiente código:

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0'

        // NOTE: Do not place your application dependencies here; they
        belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Podemos ver que el archivo nos dice que estamos utilizando el repositorio jcenter(), este repositorio es el que usa Maven, en las siguientes líneas vemos que se tiene una dependencia, la dependencia es simplemente el plugin de Gradle para Android Studio, estamos utilizando la versión 2.2 como marca esa dependencia. Las otras instrucciones por ahora no son relevantes para nuestro proyecto.

build.gradle (Module)

El código de este archivo es el siguiente:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.0"
    defaultConfig {
        applicationId "studio.android.libro.josedlujan.com.holamundo"
        minSdkVersion 22
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
            "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-
annotations'
    })
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

La primera instrucción dentro del archivo es simplemente la aplicación del plugin correspondiente.

```
apply plugin: 'com.android.application'
```

Ahora tenemos el bloque de Android en el que se especifica la versión de las herramientas y del SDK que utilizamos para la compilación, este se lo proporcionamos cuando creamos el proyecto en el capítulo del “Hola Mundo”.

Aparece también la versión mínima que utilizamos, el target, la versión de código y version name. Todos estos son los valores por defecto que se generan al crear el proyecto.

El ***minSdkVersion*** nos indica la versión mínima de SDK con la que debería de funcionar la aplicación, cuando creamos el proyecto lo indicamos, aunque se puede cambiar manualmente borrando y colocando otro número de SDK.

El ***targetSdkVersion***. Valor entero que designa el nivel de API al cual se dirige la aplicación. Si no se configura, el valor predeterminado es igual al valor asignado a la ***minSdkVersion***.

El ***VersionCode***. Este es el número de versión del código de la aplicación, nos sirve para saber con qué versión estamos trabajando.

La propiedad destacada en negrita denominada ***VersionName*** es para el desarrollador o el equipo que desarrolló la aplicación, y *VersionName* escrito en cursiva es para el público en general.

Finalmente tenemos el bloque de dependencias que nosotros podemos agregar, es recurrente en un proyecto de Android agregar algunas de las dependencias, algunas del mismo Google y otras de terceros que nos permiten utilizar y aprovechar librerías ya creadas y probadas por la comunidad de desarrolladores Android.

6 ACTIVITY

Si creamos una pantalla para una aplicación, podemos decir técnicamente que estamos creando una actividad, si nuestra aplicación tiene 10 pantallas al final de hacerla diremos que tiene 10 actividades.

Ya con un poco de experiencia el desarrollador entenderá que esto puede variar porque hay otras formas de crear pantallas, pero lo que siempre va a ser igual es que una actividad es la forma básica, aunque existen otras formas como:

- Fragments
- AppCompatActivity
- Y otras

Recordemos que Java es la base de una aplicación nativa y una de las principales características de Java es que casi todo viene dentro en una clase o es una clase, en Android esto no tendría por qué cambiar. La clase básica lleva por nombre “Activity”, cuando vamos a crear una pantalla hacemos que esta herede de “Activity” que es una clase ya definida por Android y así solo reutilizamos y modificamos a nuestro gusto. Entonces si creamos “Pantalla UNO” y queremos que esta se vuelva una actividad, lo que hacemos es marcar una herencia de la clase “Activity” para que se comporte de esta forma y la podamos utilizar como tal.

Si seguiste el libro hasta este capítulo ya tenemos un “Hola Mundo” creado.

En el directorio donde indicamos que existen los archivos Java, el directorio Java, contiene ya un archivo que es MainActivity.java; si hacemos doble clic vamos a poder ver la estructura básica de una activity hoy en día. Este es el código que contiene:

```
package studio.android.libro.josedlujan.com.holamundo;

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

public class MainActivity extends AppCompatActivity {

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

La primera línea es únicamente el paquete del proyecto.

```
package studio.android.libro.josedlujan.com.holamundo;
```

Las siguientes líneas son los dos imports básicos que necesitamos: el de la Clase que permite crear el Activity y el Bundle.

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

La siguiente línea es la declaración de una clase con el nombre MainActivity y la cual heredará de la clase AppCompatActivity, hoy en día la clase que se utiliza para crear una activity es esta, esto se debe a que las actividades han ido evolucionando con el paso del tiempo y también con la implementación de nuevas funcionalidades, animación o efectos, por eso es que la actividad que se usaba en un principio ha ido evolucionando hasta llegar a lo que hoy conocemos como AppCompatActivity, aun puedes utilizar como base la clase Activity sin problemas, pero te estarías perdiendo la animación, los efectos y algunos métodos que te ayudarán a crear una mejor aplicación.

El siguiente bloque de código dentro del archivo MainActivity.java es el método onCreate, las clases vienen por defecto con algunos métodos que podemos utilizar si los necesitamos, en el caso de las activities siempre vamos a necesitar al menos un método de estos, el método es el onCreate ya que en él le indicamos a la actividad cuál va a ser el archivo XML que va a desplegarse en la pantalla cuando la activity sea llamada. La instrucción que se encarga de eso específicamente es la instrucción:

```
setContentView(R.layout.activity_main);
```

Como se puede ver le estamos pasando entre los paréntesis la indicación del archivo, que es el archivo activity_main, así que, si el desarrollador crea otra interfaz para la activity, simplemente tiene que venir a esta instrucción y cambiar el nombre de la activity por el de la nueva y listo.

El método onCreate es un método que se ejecuta siempre que se crea la activity, así que siempre que estemos creando la activity MainActivity.java vamos a estar desplegando el archivo XML que coloquemos en esta instrucción.

Ciclo de vida de una activity

Las activities tienen un ciclo de vida:

- Se crean
- Se inician
- Se pausan
- Se reinician
- Se detienen
- Se destruyen

Es importante conocer el ciclo de vida de las actividades ya que podemos aprovecharlo durante el desarrollo de nuestra aplicación y además es necesario para ejecutar ciertas instrucciones dependiendo de la lógica de nuestra aplicación.

Dentro del ciclo de vida de la activity debemos saber que el ciclo de vida de una actividad tiene estados y al pasar de un estado a otros tenemos métodos predefinidos a los que nosotros les podemos colocar instrucciones para ejecutarse en ese momento.

El ejemplo más sencillo es cuando se inicia una actividad, en ese momento de creación pasa por el método *onCreate()*, es decir, antes de crear la actividad pasamos al método y ejecutamos lo que dice, después de esto decimos que la actividad está creada.

El estado después de pasar por el *onCreate()* sería “creada”. Así como el método *onCreate()* tenemos una lista de métodos a los cuales podemos acceder y modificar o colocar instrucciones dependiendo de nuestras necesidades:

- *onCreate()*
- *onStart()*
- *onResume()*
- *onPause()*
- *onStop*
- *onRestart()*
- *onDestroy()*

onCreate()

Es el que debemos ejecutar siempre al inicio de una actividad, en este definimos por ejemplo la interfaz de la actividad, las variables, etc. Este método por lógica solo se ejecuta una vez, en el momento de invocar o llamar a la actividad. También en él vamos a encontrar en la mayoría de las ocasiones cómo se define un archivo XML como la parte gráfica de la actividad o también la configuración de la interfaz.

Cuando el método *onCreate()* termina de ejecutarse llama al método *onStart()* seguido de *onResume()*, esto sucede de manera muy rápida.

La actividad se vuelve visible para el usuario cuando llamamos al método *onStart()* pero como sigue muy rápido el *onResume()*, en este *onResume()* permanece la actividad hasta que sucede “algo” con esta actividad.

onStart()

Aquí es donde la actividad se comienza a mostrar al usuario.

onResume()

Aquí la actividad entra en primer plano y el usuario interactúa con la actividad, la palabra con la que podríamos traducir este sería “corriendo” o “ejecutando”.

onPaused()

Aquí se encuentra parcialmente oscurecida por una actividad que se halla en primer plano; por ejemplo, está medio transparente o no cubre toda la pantalla, en este estado no se reciben datos de entrada del usuario y no puede ejecutarse código.

onStop()

En este método se encuentra completamente invisible u oculto para el usuario, podemos decir que se encuentra en el “fondo” o que se congela. Por ejemplo, las variables e información se mantienen pero no podemos ejecutar el código.

onRestart()

Este método se llama después del *onStop()* cuando la actividad actual se está volviendo a mostrar al usuario, es decir, cuando se regresa a la actividad. Después, de este continúa el *onStart()*, luego el *onResume()* y finalmente ya está de nuevo mostrándose la actividad al usuario.

onDestroy()

Cuando el sistema destruye la actividad, se manda llamar al método *onDestroy* para la actividad. Este método es la última oportunidad que tenemos de limpiar los recursos y si no los eliminamos podrían tener un mal efecto en el rendimiento para el usuario. Es buena práctica asegurarse de que los hilos que creamos sean destruidos y las acciones de larga duración también sean detenidas.

7 LAYOUTS

Todas las interfaces gráficas de una aplicación Android parten de una base parecida a la de un sitio web, en los sitios web desarrollamos una interfaz gráfica gracias a la tecnología HTML, esta es la que nos permite con relación a sus etiquetas ir estableciendo los elementos necesarios, la estructura y asignando diferentes valores a los atributos para finalmente lograr la interfaz que necesitamos.

En el caso de Android reemplazamos el HTML por XML, conocido en el mundo del desarrollo como un primo muy cercano a HTML ya que ambos trabajan con etiquetas y además XML en su momento fue un lenguaje de etiqueta pensando en desarrollar sitios web, aunque su manejo estricto fue algo que no permitió su implementación, en el caso de Android es de lo mejor que se pudo hacer ya que esa forma rígida y estricta permite evitar errores en las aplicaciones.

Todos los archivos que tienen una interfaz definida están con la extensión .XML, como vimos en el capítulo de la estructura de una aplicación, estos archivos los vamos a encontrar almacenados en el directorio:

`res/layout`

Este directorio es el asignado por Android para gestionar los archivos que van a ser interfaces de nuestra aplicación. Como se puede imaginar el lector de ahí proviene el nombre de este capítulo.

Además de que definimos la interfaz gráfica de una aplicación también colocamos una estructura y orden a nuestros elementos, podemos decir que

definimos contenedores, estructura y orden al mismo tiempo que le damos forma a una interfaz gráfica.

Antes de definir cualquier interfaz gráfica y colocar elementos o entradas de texto debemos de entender que hay un elemento superior que estudiar, este elemento es el layout.

Un layout es un contenedor que nos permite asignar ciertas propiedades o características a los elementos que se colocarán dentro de los layouts.

Tenemos diferentes tipos de layouts que nos van a permitir distintos acomodos a los elementos interiores.

Explicaré con un ejemplo qué es un layout.

Imaginemos que tenemos una casa con 5 cuartos. Cada uno de los cuartos tiene ciertas reglas, por ejemplo.

- El primer cuarto guarda todos los elementos, pero les coloca un tamaño de 20 cm x 20 cm
- El segundo cuarto guarda todo lo que se coloque dentro de izquierda a derecha.
- El tercer cuarto guarda todo lo que se coloque dentro del de arriba del elemento anterior.
- El cuarto guarda todo lo que se coloque dentro de él a la izquierda del elemento anterior
- El quinto cuarto guarda todo lo que se coloque dentro de él a la derecha del elemento anterior.

Así que sin importar lo que uno coloque en el cuarto por arte de magia, se van a tener que seguir esas reglas, decimos por “arte de magia” ya que, aunque la persona entre al cuarto y le asigne una posición al elemento, este se moverá a la posición donde la lógica del cuarto diga.

Los layouts tienen ese funcionamiento, por eso es que cada uno tiene una lógica diferente, funcionamiento y propiedades que lo hacen más adecuado para cada una de las situaciones a las que nos vamos a afrontar.

El lector debe de tener claro que no existe el mejor layout, el peor layout, el más sencillo, el más difícil, el más complejo, simplemente existe el más adecuado para cada una de las situaciones a las que nos enfrentaremos, pero además lo más curioso de todo es que una interfaz gráfica se puede hacer con solamente un tipo de layout o con otro layout, además se pueden combinar uno con otro elemento, podemos decir que es por cuestión de gustos del desarrollador la selección de los layouts, pero también es cierto que conforme un desarrollador obtiene más experiencia en la implementación de elementos visuales, este va a poder decidir qué es lo más conveniente dependiendo del tiempo, situaciones y/o beneficio del usuario.

Tipos de layouts

Debemos de visualizar cualquier layout como un contenedor que contiene reglas para colocar elementos de una interfaz gráfica de Android.

Los layouts elementales que tenemos en Android son:

- `LinearLayout`
- `RelativeLayout`
- `AbsoluteLayout`
- `TableLayout`
- `FrameLayout`

Los más usados en el desarrollo de aplicaciones Android son:

- `LinearLayout`
- `RelativeLayout`
- `FrameLayout`

Esto no quiere decir que no se usen los otros layouts, simplemente por las condiciones y las propiedades que tienen estos tres últimos son a los que más recurren los desarrolladores de aplicaciones Android en el momento de crear una aplicación.

LinearLayout

Este elemento es el más fácil de entender de todos los elementos layouts que tenemos ya que su propiedad principal es muy sencilla. El acomodo dentro del linearlayout puede ser de dos formas: de forma vertical o de forma horizontal.

Si el desarrollador selecciona la forma vertical, significa que el acomodo será de arriba hacia abajo, así que si ponemos 20 elementos significa que tendremos una columna de 20 elementos, es decir, uno debajo del otro. Cuando indicamos que el acomodo de este elemento va a ser de forma horizontal esto cambia totalmente, el acomodo será de izquierda a derecha, por lo tanto, tendremos 20 elementos en una misma fila, y todos los elementos se colocarán a la derecha del elemento anterior.

El LinearLayout siempre mantiene un orden, solo depende de nosotros la orientación del orden, puede ser de izquierda a derecha o de arriba hacia abajo.

Así es como se declararía un LinearLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
    android:orientation="vertical"

tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">
</LinearLayout>
```

En la primera línea como cualquier elemento tenemos la versión de XML y el encoding.

```
<?xml version="1.0" encoding="utf-8"?>
```

En la segunda línea declaramos el LinearLayout abriendo la etiqueta del mismo elemento y colocando las propiedades xmlns:android y xmlns:tools, por defecto Android Studio las coloca en el elemento padre de la pantalla, esto quiere decir que

el elemento con la jerarquía más alta va a contener siempre estas propiedades por defecto, no tendremos que preocuparnos por ponerlas.

```
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"
```

Como todo elemento que se coloca en la interfaz gráfica debe tener al menos 2 propiedades que son el alto y el ancho, estos se colocan con el respectivo `layout_width` y `layout_height`.

```
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

Finalmente tenemos la propiedad de orientación que como mencionamos es la línea más importante en un `LinearLayout` ya que gracias a esa línea el `LinearLayout` obtiene su acomodo por defecto, puede tener dos valores:

```
android:orientation="vertical"
```

```
android:orientation="horizontal"
```

RelativeLayout

Este elemento es un elemento relativo, esto quiere decir que coloca a los elementos en una posición relativa a otro elemento, dicho de una forma más sencilla es que un elemento se coloca en una posición tomando como referencia a uno o a más elementos, supongamos que tenemos primero un texto, luego un botón y finalmente un texto.

Podemos colocar el primer elemento que llamamos texto sin problemas, el segundo elemento que íbamos a colocar que es un botón lo vamos a colocar en una posición tomando como referencia el texto que ya está colocado, finalmente el tercer elemento que es un texto nuevamente lo podemos colocar tomando como referencia el texto primero o el botón, también podremos tomar de referencia ambos elementos en caso de ser necesario.

Por defecto hoy en día el “Hola Mundo” en Android utiliza como base el contenedor RelativeLayout, así que si abrimos el “Hola Mundo”, va a ser el layout que vamos a encontrar y el código será este:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"

tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

En la primera línea tenemos lo mismo que en el resto de XML:

```
<?xml version="1.0" encoding="utf-8"?>
```

El resto del código es muy parecido al LinearLayout, la diferencia más grande es que este elemento no tiene la propiedad “orientation” con la que dictamos el acomodo, cada elemento tendrá que aplicar propiedades que permitan acomodarlo dependiendo de las posiciones de otros elementos.

AbsoluteLayout

Este elemento nos permitirá colocar los elementos que iba a contener en una posición absoluta, esto quiere decir que el elemento se mantendría en esa posición sin importar lo que se suceda, por esa misma razón era un poco complejo manejarlo

ya que no se adaptaba dependiendo de las necesidades de la pantalla, la vida de este elemento fue corta, ya que salió en la API 1 y en la API 3 se consideró este layout como DEPRECATED, por lo mismo solamente lo estamos mencionando para que el lector lo conozca, aunque su implementación simplemente ya no sucede y no tiene relevancia en el mundo del desarrollo hoy en día.

Debemos dejar claro que este layout ya no está considerado dentro de las opciones que podemos utilizar cuando construimos una aplicación.

TableLayout

Este elemento se usa bastante en el momento de tener un acomodo similar al de las tablas de una hoja de cálculo, este elemento permite de manera sencilla tener un acomodo peculiar gracias a las filas y columnas.

Dentro del elemento TableLayout se debe de definir un elemento o varios TableRow dependiendo de la cantidad de filas que se quieran colocar, estos elementos contienen el tamaño de la vista de la tabla que vamos a colocar.

El código para utilizar un TableLayout sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_layout_4_open"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_open_shortcut"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```


La primera línea es como en todo XML, la segunda línea es la indicación de que vamos a abrir el elemento `TableLayout`, la siguiente línea contiene lo mismo que los elementos, la línea que es diferente es la de `stretchColumns`, esta propiedad nos permite expandir una columna para aumentar la anchura, la finalidad es ocupar todo el ancho del contenedor.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
```

A continuación, tenemos el elemento `TableRow`.

```
<TableRow>
    <TextView
        android:text="@string/table_layout_4_open"
        android:padding="3dip" />
    <TextView
        android:text="@string/table_layout_4_open_shortcut"
        android:gravity="right"
        android:padding="3dip" />
</TableRow>
```

Este elemento nos permite especificar la fila, así que si tenemos 3 `TableRow` decimos que tenemos 3 columnas, cada elemento que se encuentre dentro de un `TableRow` pertenece a esa columna ya que están formando en conjunto la vista de la fila.

FrameLayout

Este elemento junto al `LinearLayout` son los más fáciles de comprender, este elemento siempre acomodará los elementos internos en la parte superior izquierda, un error común es colocar varios elementos en este ya que uno tapará a otro, esto se debe a que el acomodo es hacia el mismo lado. Este elemento se usa comúnmente para colocar `Fragments`, otro elemento que aún no hemos visto. A diferencia de

todos los layouts que contienen propiedades diferentes, este elemento no cuenta con alguna propiedad especial, podemos decir que tiene la declaración más simple ya que no tiene propiedades que sean únicas, simplemente tiene las propiedades que cualquier otro elemento tiene.

Este es un ejemplo de un `FrameLayout` en código:

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

Este elemento como se puede ver simplemente necesita el ancho, el alto y también algunas de las propiedades por defecto dependiendo del caso de su uso, pero nos bastaría con un simple “width” y “height” para su funcionamiento.

ELEMENTOS GRÁFICOS EN ANDROID

En este capítulo vamos a hablar de los elementos que nos sirven para crear una interfaz para nuestras aplicaciones.

Primero hablaremos de algo que tenemos que saber. Los elementos visuales contienen siempre 2 propiedades, sin importar el tipo, el comportamiento, el tamaño, el color o de dónde provengan.

Esas propiedades son:

- Ancho
- Alto

Siempre van a tener estas dos propiedades, pero en inglés: Width, Height. Probablemente en el capítulo anterior te diste cuenta cuando nos referíamos a los layouts, todos contenían dos líneas de código que son las siguientes:

```
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

Se puede observar que casi siempre las propiedades van juntas, aunque no existe ningún problema al compilar si estas están separadas, pero por convención siempre encontrarás las líneas juntas y además siempre primero el ancho y luego el alto, podemos decir que es ya una convención en el mundo del desarrollo de aplicaciones Android.

En este caso los dos elementos tienen el valor: `match_parent`. Pero se pueden tener 3 valores diferentes:

- `match_parent`
- `fill_parent`
- `wrap_content`

Cuando colocamos el valor `match_parent` indicamos que el elemento va a tener el ancho de su padre, es decir, el elemento que lo contiene, por eso en el capítulo anterior hicimos énfasis en mencionar que siempre tendremos un layout como contenedor; si este contiene elementos, se vuelve el padre de todos los elementos que contiene. Por lo tanto, si un elemento coloca el valor `match_parent` en su alto o en su ancho, implica que tomará el ancho o el alto del padre al que pertenece.

El valor `fill_parent` en realidad es lo mismo que `match_parent`, aunque a partir del nivel de la API 8 (actualmente estamos en el API 24) se dejó de utilizar `fill_parent` y ahora se coloca `match_parent`, aunque se obtiene el mismo resultado, cualquier desarrollador experimentado recomendaría utilizar `match_parent`.

El valor `wrap_content` tiene un valor por decirlo de una forma “dinámico” ya que permite que el elemento coloque un tamaño dependiendo del contenido, por ejemplo: si tenemos un elemento que almacena texto y tiene en su ancho el `wrap_content`, esto implica que si tiene en el texto una cadena de texto de 10 caracteres ese será su ancho, pero si la siguiente vez la cadena de texto tiene una cadena de texto de 1000 caracteres este será el tamaño del elemento de texto. Como se puede ver en el ejemplo el tamaño tanto ancho o alto va a variar del contenido y así vamos a tener un tamaño que puede variar dependiendo siempre de las necesidades del elemento.

Estos 3 valores son los valores que puede tener por defecto un elemento. Otro atributo que pueden tener los elementos en Android sin distinción es el atributo del identificador o también conocido como “id”. El ID sirve para poder acceder desde el código Java al elemento, podemos decir que es un nombre que se le da al elemento para que accedamos al elemento y lograr interactuar con él desde el código.

El código del ID sería el siguiente:

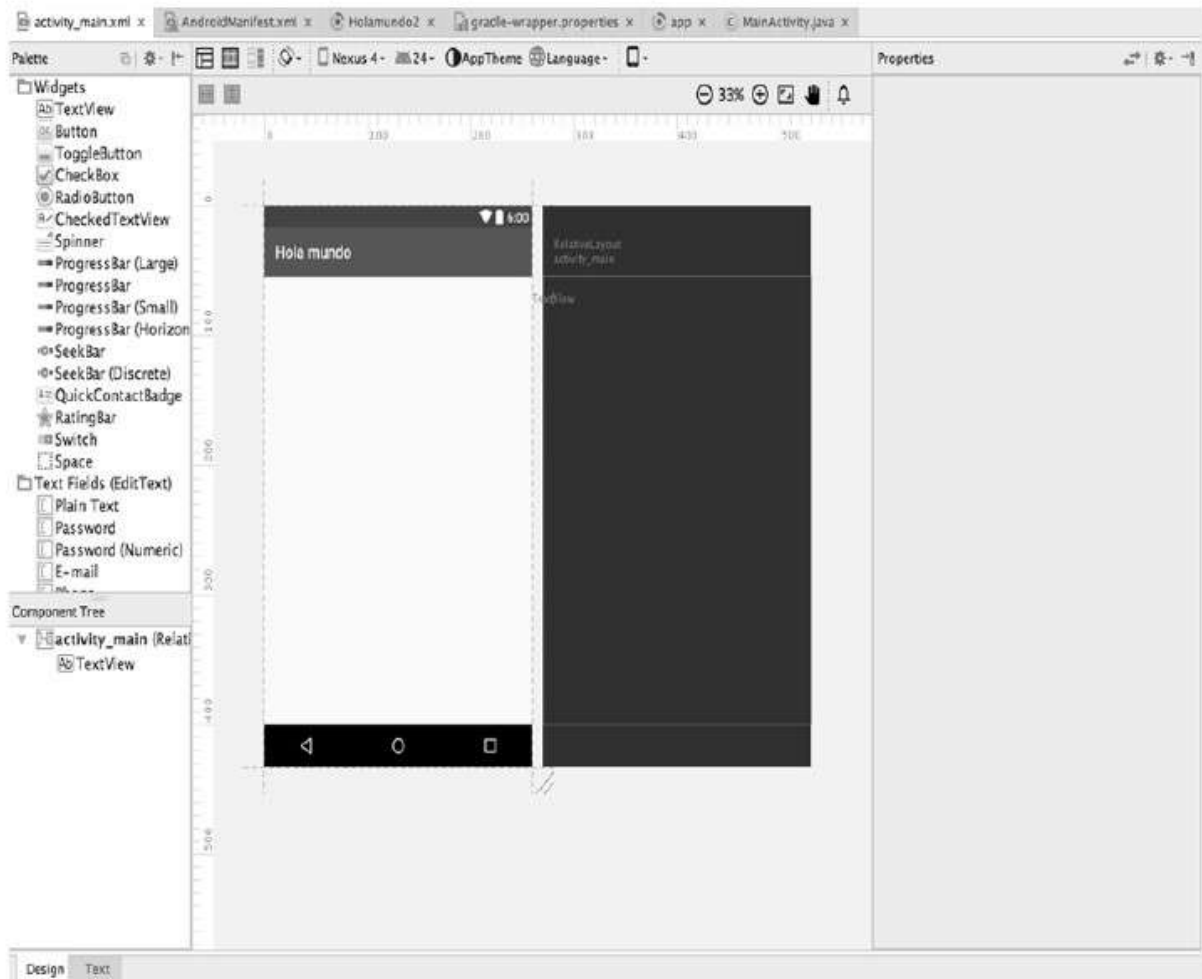
```
android:id="@+id/textView"
```

La línea anterior es lo que le podemos colocar a cualquier elemento para asignarle un identificador y desde el código Java poder acceder a este elemento más adelante.

Interfaz gráfica

Ya sabemos que los archivos con extensión XML son los que representan las vistas de nuestras actividades y estos se encuentran en el directorio res/layout. Ahora lo que vamos a hacer es abrir el archivo que se creó cuando creamos el “Hola Mundo”, es decir, el archivo MainActivity.java.

Vamos a ver la siguiente interfaz de usuario:



En la parte inferior podemos ver que tenemos dos pestañas:

- Design
- Text

Cada una de las pestañas tiene su propósito.

Pestaña “Design”

Lo que vemos es el archivo desde la interfaz gráfica, esto nos permite poder modificar el archivo de forma muy similar a una herramienta “Drag & Drop”, esto quiere decir que podremos arrastrar los elementos del lado izquierdo y soltarlos en la posición que los necesitamos, la idea es que en relación con la interfaz podamos crear nuestro diseño, esto es algo muy recurrente hoy en día en las herramientas de desarrollo de software, podemos decir que tenemos un lienzo en el que podemos arrojar los elementos que queremos dibujar.

Esta pestaña hace que Android Studio se convierta en una herramienta muy práctica y veloz, pero también tiene sus puntos malos. El diseño se puede volver dependiente de esta característica y en algunos casos en donde necesitamos modificar únicamente una propiedad o una línea del código se recurre a una herramienta visual que lo hace muy complejo ya que podríamos utilizar el texto con una simple instrucción de buscar para encontrar lo necesario.

Esta forma de trabajar se asemeja a muchas herramientas que trabajan con la filosofía WYSIWYG. Esto es el acrónimo de “What You See Is What You Get” que en español significa, lo que ves es lo que obtienes, es decir, que uno va diseñando visualmente lo que necesita y eso es lo que vas a obtener.

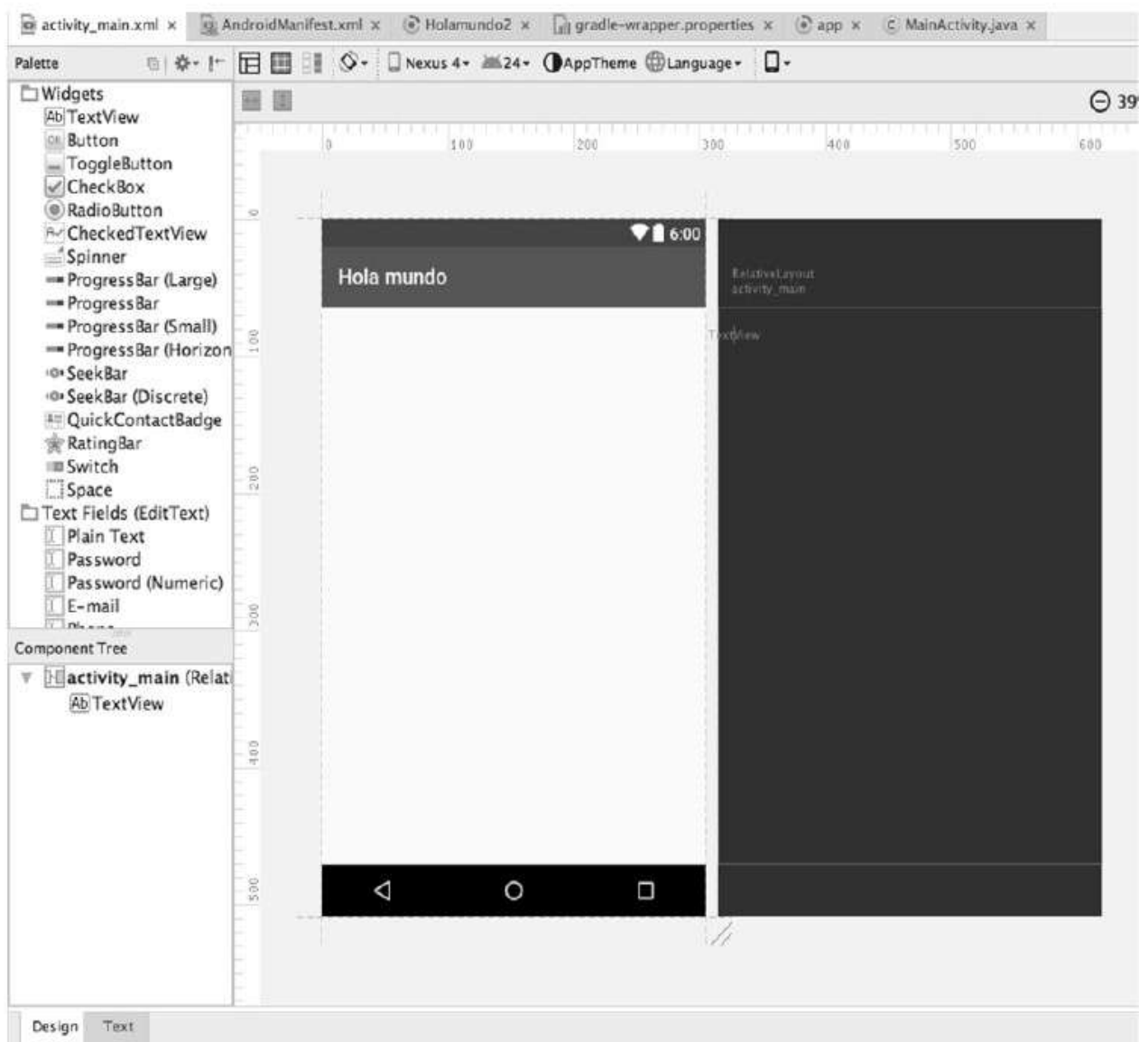
Pestaña “Text”

Esta pestaña por el contrario se basa en el código duro y puro, así que si se quiere colocar un elemento utilizando esta pestaña, lo que se va a observar es el código del archivo XML y se tendrían que agregar más líneas de código para el elemento. Este método es mucho más ligero de usar ya que no tiene que hacer el render de ninguna pantalla, pero además contiene menos bugs si lo

comparamos con la parte visual, esto se debe a la sencillez de la herramienta, ya que solamente debe de interpretar el código que el desarrollador está colocando.

A continuación, veremos 2 imágenes, la primera es para ver cómo es visualmente la opción “Design” y la segunda es para ver la opción “Text”.

Opción de diseño:



Opción de código:



Todo el código que hemos pegado hasta este capítulo del libro que tiene que ver con una interfaz gráfica es el ejemplo del uso de la pestaña “Text”.

Por sencillez utilizaremos la pestaña de texto a lo largo del curso, ya que nos permite copiar y pegar el texto en el libro sin problemas. Si tomáramos como base la parte de diseño podríamos perdernos algunas cosas visualmente importantes para el lector, además agilizará el desarrollo de una interfaz visual y por otro lado practicaremos y conoceremos los elementos visuales al tener que estar escribiendo o autocompletando las opciones.

TextView

El TextView es un elemento que nos sirve para colocar texto dentro de una aplicación, existen otras variaciones y formas, se utiliza cuando queremos colocar una cadena de texto.

Vamos a modificar nuestro main_activity.xml para que contenga un “Hola Mundo” con un LinearLayout y con un textview, el texto sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools
tools:context="studio.android.libro.josedlujan.com.holamundo.MainActiv
ity
    android:id="@+id/linearlayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
">

    <TextView
        android:text="Hello World!"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

La primera línea es como cualquier archivo XML:

```
<?xml version="1.0" encoding="utf-8"?>
```

El siguiente bloque de código tiene la definición de un layout que es el LinearLayout, en el que dejamos los elementos de configuración por defecto que colocó Android Studio que son xmlns:android y xmlns tools y context:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tolos
tolos:contet="studio.android.libro.josedlujan.com.holamundo.MainActivity"
```

La siguiente línea es la definición de un id, el nombre que le colocamos al elemento es el "linearlayout" por convención, cuando solo se tiene un elemento del mismo tipo se acostumbra dejarle el mismo nombre:

```
android:id="@+id/linearlayout"
```

Las siguientes líneas contienen la definición del alto y del ancho del linearlayout:

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Finalmente tenemos la línea más importante en el linearlayout que es la orientación, no olvidemos que esta línea es únicamente aplicable a este elemento:

```
android:orientation="vertical"
```

Dentro del linearlayout tenemos el bloque de código con la definición del textview. La primera línea de código es el atributo text, en ese atributo lo que colocamos es la cadena que texto que queremos aplicar.

```
<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Probando la orientación LinearLayout

Ya tenemos una interfaz gráfica con la que podemos comprobar el atributo de la orientación, ahora vamos a agregar 5 elementos textview que pueden ser copia sin problemas del que ya tenemos, el código quedaría así:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
```

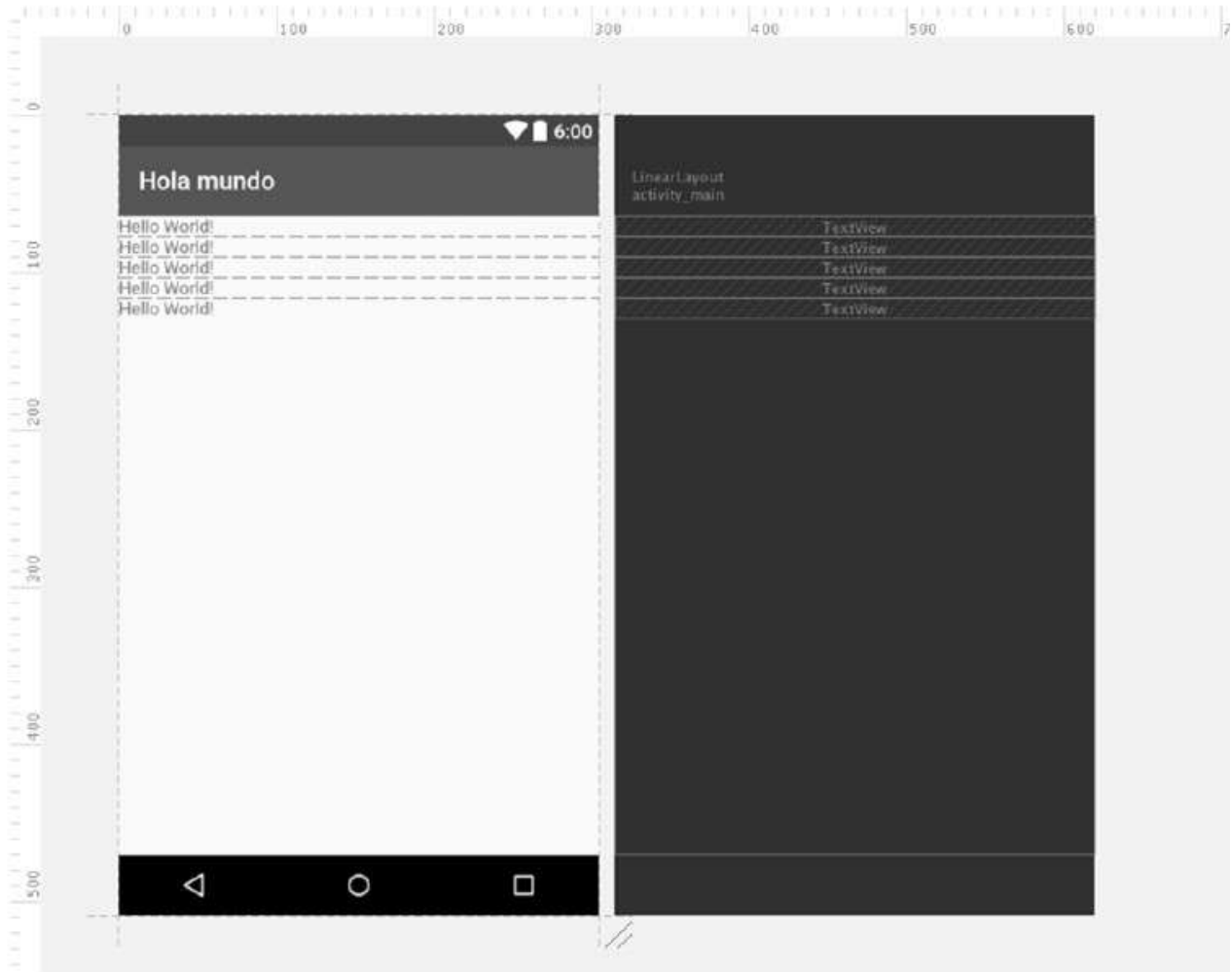
```
android:layout_height="match_parent"
android:orientation="vertical"

tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

Como podemos ver solamente agregamos 4 bloques duplicados del textview que ya teníamos para tener en total 5 elementos.

El resultado visualmente sería el siguiente:

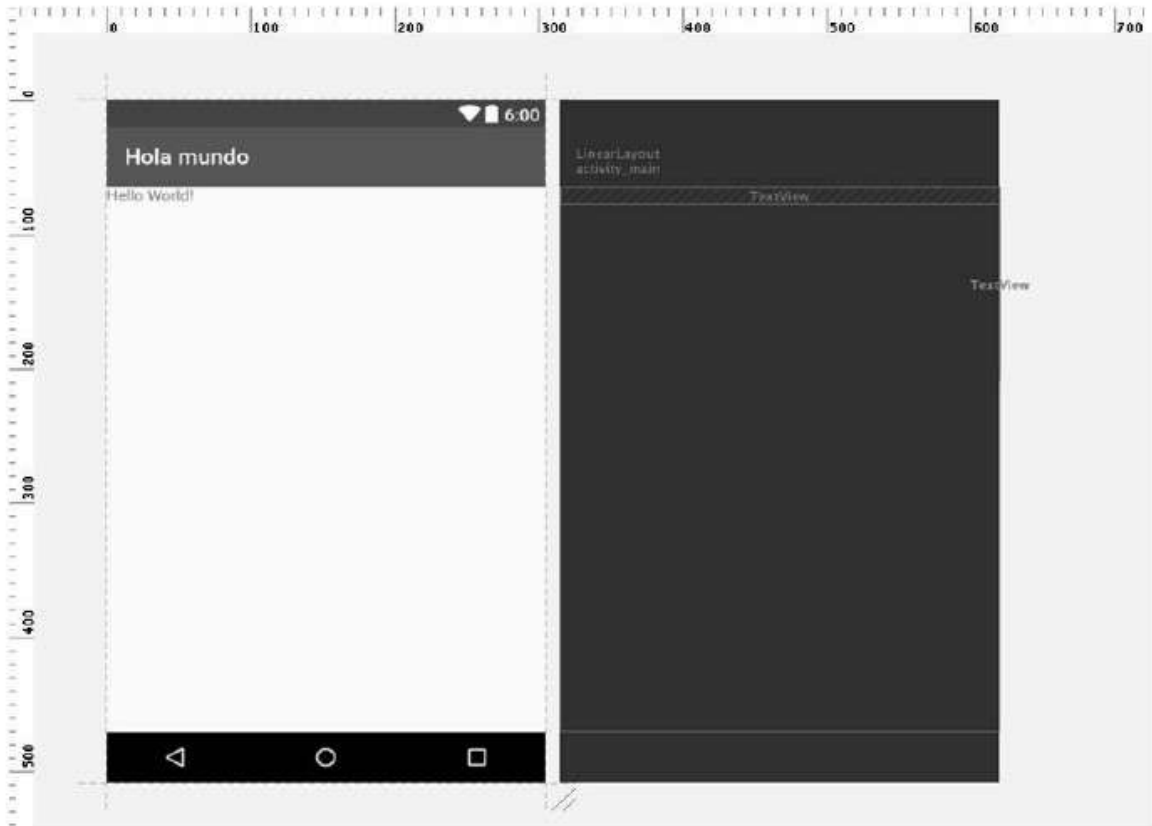


Los elementos textview están uno debajo del otro porque el contenedor padre, que es el linearlayout, colocó en su atributo orientation, que la orientación de los elementos que contendrá será de forma vertical, por lo tanto, no es importante el tipo de elemento, su tamaño o cualquier otro atributo, siempre tendrá que ir debajo del anterior.

Ahora vamos a cambiar el atributo orientation del linearlayout por horizontal:

```
android:orientation="horizontal"
```

Como podemos ver solo tocamos un atributo y al hacer el cambio el resultado visualmente será el siguiente:



Ahora solo vemos un elemento textview, esto es debido a que todos los elementos textview tienen como ancho el de su padre y su padre ocupa todo el ancho de la pantalla, los elementos sí se están dibujando, pero no los podemos ver porque abandonan la pantalla, lo que tendríamos que hacer para que se vean es colocarles un ancho ajustado, es decir, que utilizamos el valor `wrap_content` para indicarlo.

Después de los cambios de orientación y del ancho, el código sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"

    tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

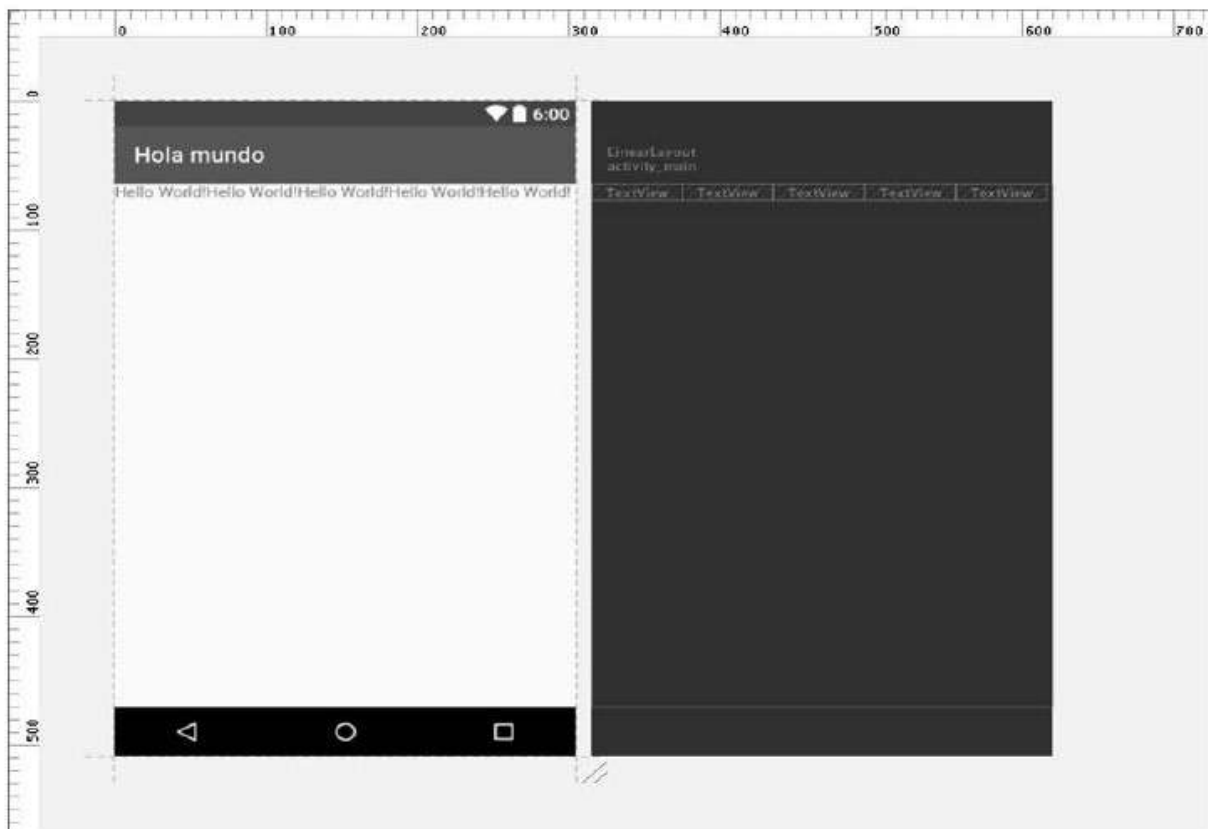
    <TextView
        android:text="Hello World!"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>

```

El resultado visualmente del código anterior sería:



Ahora sí podemos ver los elementos textview gracias a la modificación del ancho del textview y también observar el efecto que produce la orientación dentro de los elementos hijos.

Propiedades del TextView

Todos los elementos tienen distintas propiedades y esto es natural ya que cada uno tiene un funcionamiento determinado, así que a lo largo de este capítulo cuando mencionemos algún elemento también daremos un repaso rápido a sus propiedades, aunque no consultaremos todas ya que no se usan siempre, dependiendo de las necesidades de cada aplicación.

Tomando como referencia el ejemplo anterior, podemos quedarnos únicamente con un elemento textview para este apartado.

textSize

Una de las propiedades más comunes en un textview es la propiedad del tamaño, esta propiedad se llama textSize y permite asignarle un tamaño diferente al texto de este elemento, así quedaría el código de un textview con un cambio de fuente a 20sp:

```
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
/>
```

En Android no se utiliza como medida el píxel, que es a lo que uno está acostumbrado en casi cualquier tecnología, en el mundo Android podríamos decir que esa medida nunca se debería de usar, ya que si se usa por la naturaleza de los dispositivos, uno estaría creando aplicaciones de forma incorrecta y tendría muchos problemas con el diseño de Android, así que para las fuentes utilizamos la medida SP.

SP significa Scale-independent Pixels, así que por ahora nos quedaremos con esta explicación ya que más adelante tendremos un capítulo en donde analizaremos el tema a fondo.

textColor

Esta propiedad nos permite cambiar el color del texto de nuestro textview, el color se puede dar de varias formas, una de las más comunes es utilizar el color en el formato hexadecimal como se hace en un sitio web con CSS y HTML.

El código sería el siguiente:

```
<TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:textColor="#FF0000"
/>
```

El texto que se encuentre en ese textview tendrá el color rojo, ya que ese valor en hexadecimal representa el color rojo.

Textcolor utilizando el archivo colors.xml

Otra forma de asignar colores no solamente al textview sino a cualquier elemento es utilizar el archivo colors.xml que se encuentra dentro del directorio:

res/values/colors.xml

En este directorio se acostumbra a almacenar los colores y asignarles un nombre para poder así acceder a ellos desde cualquier parte dentro de la aplicación Android pero además permite reutilizar el código y no tener que estar definiendo el color cada ocasión que este sea utilizado.

Si abrimos el archivo colors.xml, veremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```


Tenemos por defecto la definición de 3 colores, esto se debe a que es una pauta que se marcó hace unos años con la aparición de material design. Podemos utilizar alguno de esos colores o podemos agregar los propios, lo que haremos es agregar el color rojo para usarlo.

Agregaremos el color rojo con su respectivo formato hexadecimal, así que el archivo ahora quedará con un elemento más de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
  <color name="red">#FF0000</color>
</resources>
```

Para utilizar el color que acabamos de definir que es el color rojo, pero le colocamos el nombre en inglés “red” para que llevemos la misma pauta respecto a los nombres de los colores, tenemos que ir al elemento que va a utilizar ese color e indicarle que vamos a utilizar el archivo colors para tomar el color. El código es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal"

  tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

  <TextView
    android:text="Hello World!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:textColor="@color/red"
  />

</LinearLayout>
```

Podemos observar que la línea que teníamos antes:

```
android:textColor="#FF0000"
```

Simplemente cambió a esta:

```
android:textColor="@color/red"
```

Estas serían las dos formas de acceder a los colores dentro de nuestra aplicación Android, no únicamente para el texto.

Button

El botón es uno de los elementos más usados dentro de una aplicación Android ya que nos permite recibir información por el usuario de una manera rápida.

Existen elementos como las imágenes que se pueden comportar como un botón, aunque tienen propiedades diferentes, el botón por lógica es el elemento que se debe de utilizar como una entrada de información para que el usuario indique la siguiente instrucción a ejecutar.

El código anterior es el de un botón dentro de un linearlayout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"

    tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

    <Button
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

El primer bloque de código es un linearlayout con su configuración adecuada, además contiene una orientación horizontal para ir colocando los elementos uno a lado de otro en caso de ir agregando más.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal"

  tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">
```

El segundo bloque de código es el elemento botón en el que se coloca un texto al botón que es "Hello World!". Como vemos esta propiedad la comparte con el textview, como todo elemento contiene un ancho y un alto que en este caso las dos características son wrap_content.

```
<Button
  android:text="Hello World!"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```

Como normalmente el botón utiliza la lógica de la programación para crear los comportamientos necesarios y poder ejecutar algo, cuando se le hace clic necesitará un identificador, así que le podríamos agregar un id para poder utilizarlo o llamarlo desde el código Java:

```
<Button

  android:id="@+id/boton"
  android:text="Hello World!"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
/>
```

El id que se le colocó es el id= boton, como en este caso solo tenemos un botón no existe problema con el nombre, pero cuando tenemos dos o más botones no se recomienda este tipo de nombre, el identificador podría ser la acción que vamos a ejecutar cuando se le haga clic, por ejemplo, si al dar clic vamos a hacer un inicio de sesión se podría llamar:

```
android:id="@+id/iniciar_sesion"
```

O también lo podríamos llamar:

```
android:id="@+id/login"
```

Depende de qué pauta queremos seguir para nombrar las variables, pautas en inglés o en español, por recomendación el lector debería de considerar como primera opción la pauta en inglés.

EditText

El editText entraría entre los 3 elementos más usados en Android junto con el botón y el textView. Este elemento nos permite escribir datos para que la aplicación los pueda utilizar, leer o considerar dependiendo del caso.

Este tipo de campo es también de los más viejos en el desarrollo de Android, existe desde los inicios de Android, la mayoría de las aplicaciones hoy en día cuenta con él.

En el siguiente código vamos a colocar un editText dentro de un linearlayout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"

    tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

    <EditText
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"  
    />  
  
</LinearLayout>
```

El primer bloque es repetitivo del elemento del linearlayout, el segundo bloque contiene el elemento editText y como vemos solo contiene 2 atributos, el ancho y el alto; si vemos la pantalla de la aplicación en este momento no veríamos nada, estaría en blanco ya que no tiene algo como un punto de referencia visual.

El editText cuenta con la propiedad de texto para que podamos colocarle algún texto:

```
<EditText  
  
    android:text="Hello World!"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
/>
```

El detalle en este caso es que cuando queremos escribir en el editText tendremos que borrar el texto y esto es muy tedioso para el usuario ya que si tenemos una cadena de texto que diga: “Introduce tu username” o “Username” algo habitual que se coloca para indicar al usuario qué tipo de información debe incorporar lo vuelve muy complejo.

Por otro lado tenemos una propiedad que es la que comúnmente se utiliza, que es hint, esta propiedad les debe de sonar familiar a los desarrolladores que vienen del mundo web y que trabajan con HTML5 ya que en el momento de seleccionar el elemento editText se borra automáticamente el texto que contiene para esperar que el usuario lo añada, esto evita que el usuario tenga que borrarlo manualmente y le indica de manera correcta el tipo de dato que se espera introduzca, podemos decir que esta es la mejor opción para notificarle qué es lo que debe de hacer.

Este sería el código:

```
<EditText
```

```
    android:hint="Hello World!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
/>
```

Practicando RelativeLayout

En este capítulo ya practicamos el acomodo de elementos utilizando el LinearLayout, ahora vamos a dar un repaso a cómo se hace el empleo de elementos con RelativeLayout.

Ya mencionamos con anterioridad que el acomodo dependerá de la posición de otro elemento, primero colocaremos un RelativeLayout vacío:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="studio.android.libro.josedlujan.com.holamundo.MainActivity">

</RelativeLayout>
```

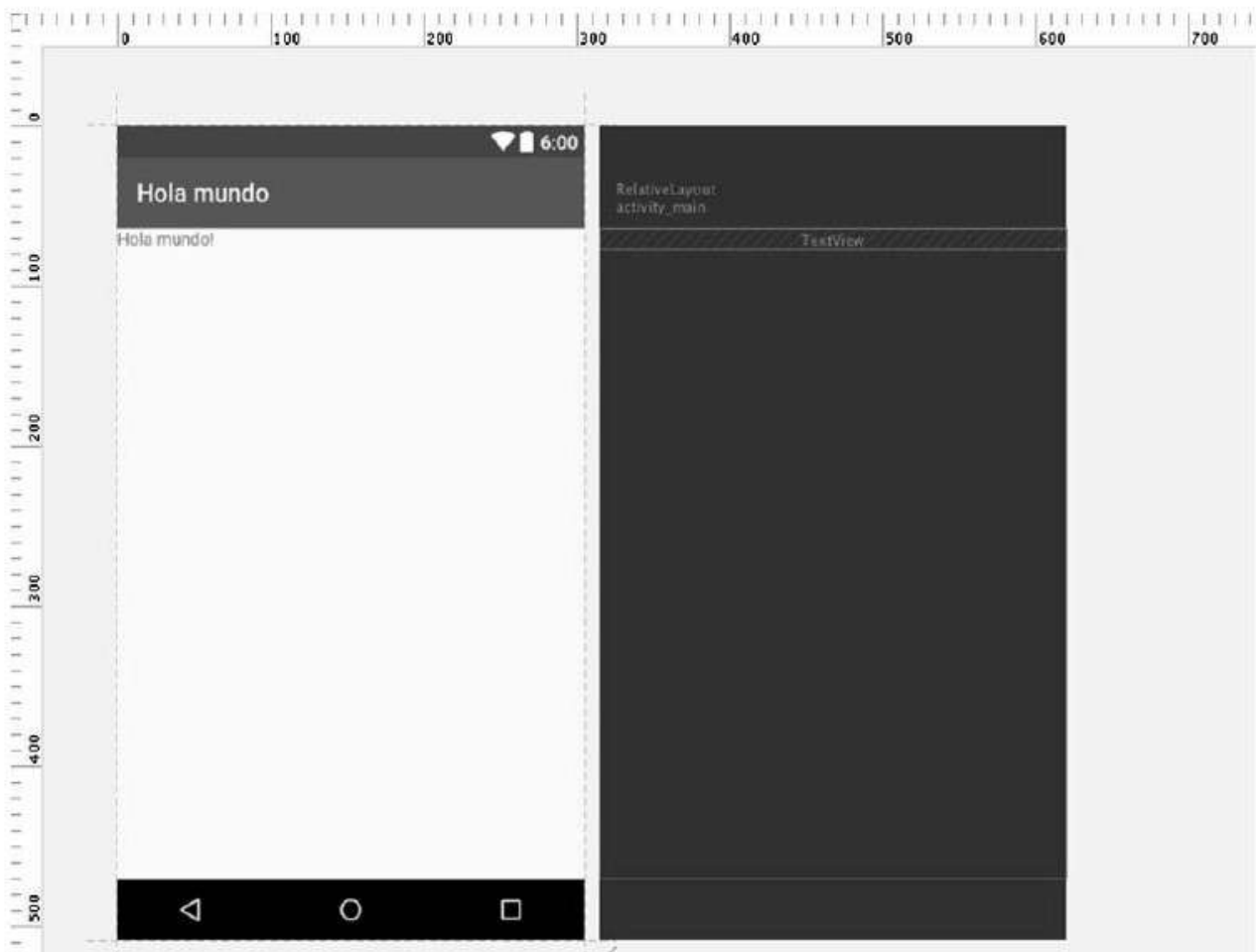
Si revisamos el código podemos ver que no contiene nada fuera de lo común, solamente la configuración inicial como cualquier layout y el ancho y el alto.

Ahora vamos a colocar el primer elemento que será un textview que contendrá la frase “Hola mundo!”, el código quedaría de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="studio.android.libro.josedlujan.com.holamundo.MainActivity">
```

```
<TextView  
    android:text="Hola mundo!"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
/>  
  
</RelativeLayout>
```

El bloque del textview es la definición más básica que podemos tener de este, solo contiene texto, un ancho y alto. Podemos ver la representación gráfica de la siguiente manera:



En la imagen podemos ver que el elemento se encuentra alojado en la parte superior izquierda por defecto. Ahora vamos a agregar un botón que diga el texto “Enviar”; el código sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

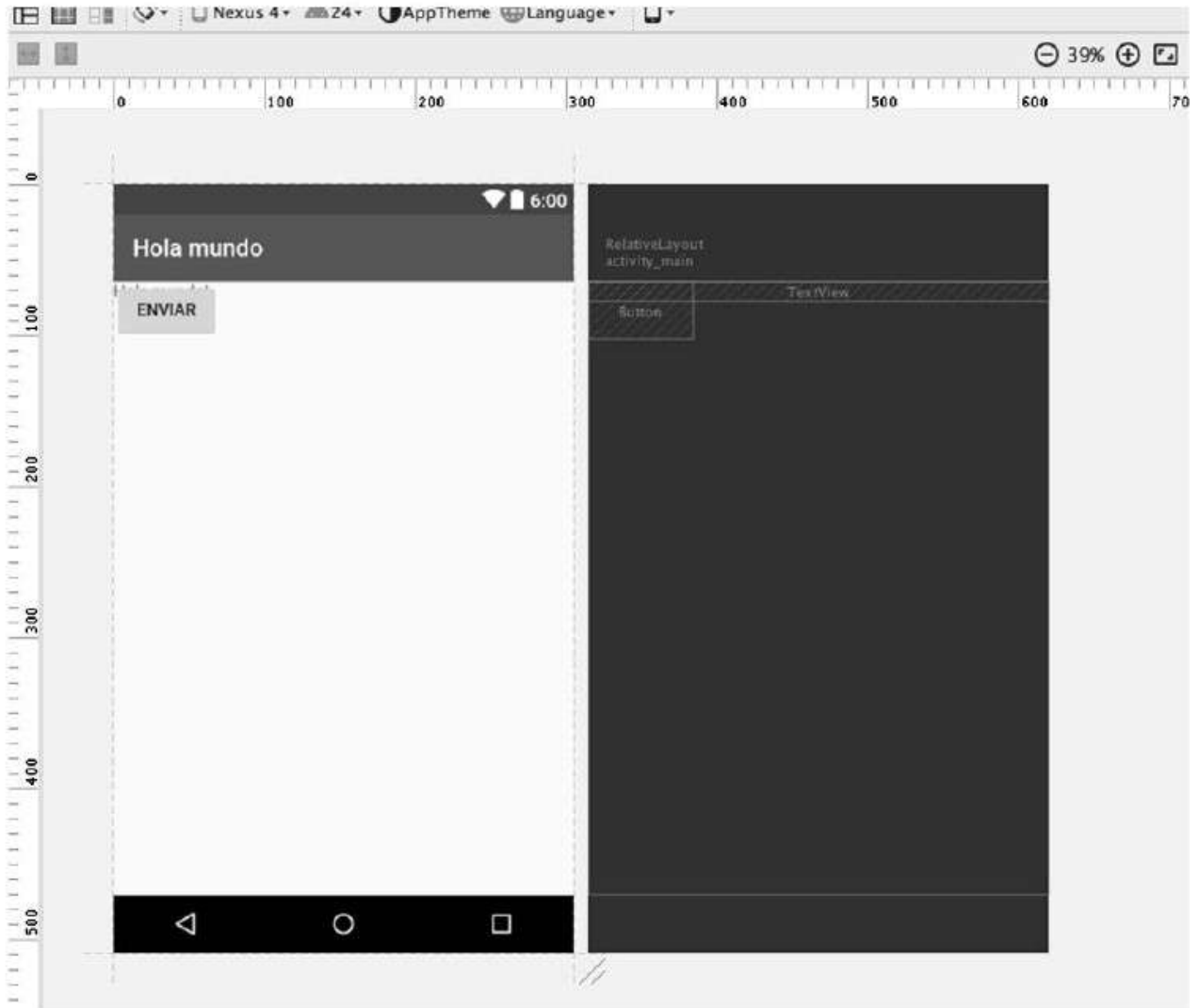
    <TextView
        android:text="Hola mundo!"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />
    <Button
        android:text="Enviar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RelativeLayout>
```

Hemos agregado un botón después del bloque de texto en el que solo colocamos el texto “Enviar” y además colocamos su ancho y alto:

```
<Button
    android:text="Enviar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Si vemos la pestaña de diseño, podremos observar la siguiente imagen:



Si observamos la imagen ahora vemos solamente el botón o una muy pequeña parte del texto que se encuentra detrás del botón con el texto “Enviar”. Esto es porque por naturaleza los elementos se van acomodando de esta forma ya que no cuentan con una orden de acomodo, para que este no pase debemos de asignarle una posición al elemento y así lograr un acomodo correcto.

Lo que haremos es modificar los dos elementos, el elemento botón y el elemento textview:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
```

```

    android:layout_height="match_parent"

    tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

    <TextView
        android:id="@+id/textview"
        android:text="Hola mundo!"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />
    <Button
        android:text="Enviar"
        android:layout_below="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RelativeLayout>

```

La modificación que le hicimos al textview es únicamente la de colocarle un id para poder utilizarlo como referencia de otros elementos con esta línea al principio de la definición de sus elementos:

```

    android:id="@+id/textview"

```

En el caso del botón agregamos una línea que nos permite colocarlo después de un elemento, esto es gracias al atributo layout_below que sirve para este fin, le tenemos que indicar el id del elemento al que tomaremos como referencia, la línea quedaría de la siguiente forma:

```

    android:layout_below="@+id/textview"

```

Con estas dos líneas en cada uno de los elementos estamos realizando el acomodo que nos permite utilizar el elemento relativelayout, así se verían los elementos en la pestaña de diseño:

- Adaptador (el que se encarga de rellenar la lista) (Archivo FrutasVerdurasAdapter.java)
- Clase de los elementos que colocaremos en la lista. *Opcional (Archivo FrutasVerduras.java)

El archivo activity_main tiene este código:

```
<LinearLayout
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"
    android:orientation="vertical"
    tools:context="com.example.ejemplolista.MainActivity" >

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

    </ListView>

</LinearLayout>
```

Solo contiene un ListView con un ancho, alto y un id.

Para crear los siguientes layouts pulsamos con el botón derecho en el directorio layout, seleccionamos nuevo archivo y le ponemos el nombre que necesitamos.

El archivo list_header_row.xml lo creamos en el directorio layout como cualquier layout, este es el código que contiene:

```
<?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="wrap_content"
    android:orientation="vertical" >
```

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/encabezado"
    android:textSize="25sp"
    android:background="#234562"
    android:textColor="#ffffff"
    android:gravity="center_horizontal"
/>

</LinearLayout>

```

El código anterior es para definir el encabezado de nuestra lista, le colocamos ciertos atributos y valores como el de texto, tamaño, fondo, color del texto y la gravedad para el acomodo del texto.

Creamos un nuevo archivo que es `listview_item_row.xml`, el código es:

```

<?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="wrap_content"
    android:orientation="horizontal"
    android:weightSum="10"
    >

    <ImageView
        android:id="@+id/imagen"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"

        />
    <TextView
        android:id="@+id/tv"

```

```

        android:layout_width="0dp"
        android:layout_weight="7"
        android:layout_height="match_parent"
        android:text="@string/texto"
        android:textColor="#00FF00"
        android:gravity="center_vertical"
    
```

```

    />

```

```

</LinearLayout>

```

Tenemos un `LinearLayout` y una orientación de forma horizontal y con un peso total de “10” para distribución de los elementos. Dentro del `LinearLayout` colocamos un `ImageView` y le asignamos un peso de “3”, la imagen que le vamos a colocar es la que viene por defecto en Eclipse que es el “`ic_launcher`” y creamos el segundo elemento que es un `TextView` al que le ponemos el peso de “7”, de texto le podemos poner lo que sea, en este caso usamos una cadena que se llama “`texto`” que se encuentra en el archivo “`strings.xml`”.

Ahora creamos un archivo Java, este archivo lo llamaremos `FrutasVerduras.Java`, el código es este:

```

package com.example.ejemplolista;

public class FrutasVerduras {

    public int icon;
    public String title;
    public FrutasVerduras(){
        super();
    }

    public FrutasVerduras(int icon, String title) {
        super();
        this.icon = icon;
        this.title = title;
    }
}

```


Colocamos dentro de la clase 2 atributos, un icono del tipo entero que es la imagen que queremos colocar a cada elemento de la lista, además de una cadena de texto que se llama *title*:

```
public class FrutasVerduras {

    public int icon;
    public String title;
```

Definimos un constructor en caso de querer crear un elemento y que este no reciba parámetros, para eso usamos el *super()* que hace referencia al constructor de la superclase:

```
public FrutasVerduras(){
    super();
}
```

El *super()* es un tema específico de Java, en caso de no saber de qué trata recomiendo leer un poco sobre el tema en algún libro de Java para principiantes ya que conforme se va adentrando en el desarrollo de Android se tienen que ir creando o utilizando constructores.

Finalmente creamos el caso para cuando recibimos la imagen y el título como parámetros en el momento de crear el elemento. Utilizamos *super()* pero además le asignamos la imagen y el título que recibimos a nuestro elemento.

```
public FrutasVerduras(int icon, String title) {
    super();
    this.icon = icon;
    this.title = title;
}
```

Ahora crearemos un adaptador, este va a ser para poder llenar la lista con la información. Lo podemos ver como un puente y nos permite asignar una colección de datos a un elemento. Ese es el principio de un adaptador.

El archivo lo llamaremos FrutasVerdurasAdapter.java y el código es el siguiente:

```
package com.example.ejemplolista;

import android.app.Activity;
```

```

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class FrutasVerdurasAdapter extends
ArrayAdapter<FrutasVerduras>{
    Context context;
    int layoutResourceId;
    FrutasVerduras data[] = null;

    public FrutasVerdurasAdapter(Context context, int
layoutResourceId,
        FrutasVerduras[] data) {
        super(context, layoutResourceId, data);
        // TODO Auto-generated constructor stub
        this.context = context;
        this.layoutResourceId = layoutResourceId;
        this.data = data;
    }

    public View getView(int position, View convertView,
ViewGroup parent) {

        View convertView;
        FrutasVerdurasHolder holder = null;

        if(row==null){
            LayoutInflater inflater =
((Activity)context).getLayoutInflater();
            row = inflater.inflate(layoutResourceId, parent, false);

            holder = new FrutasVerdurasHolder();
            holder.imagen =
(ImageView)row.findViewById(R.id.imagen);
            holder.texto = (TextView)row.findViewById(R.id.tv);
            row.setTag(holder);

```

```

        }else{
            holder=(FrutasVerdurasHolder)row.getTag();
        }

        FrutasVerduras frutasVerduras = data[position];
        holder.texto.setText(frutasVerduras.title);
        holder.imagen.setImageResource(frutasVerduras.icon);

        return row;
    }

    static class FrutasVerdurasHolder{
        ImageView imagen;
        TextView texto;
    }
}

```

Hacemos la herencia de una clase base que tenemos que se llama `ArrayAdapter`. Dentro de las “< >” colocamos *FrutasVerduras* que es el “tipo” de datos que va a contener y manejar este adaptador. Si hacemos memoria, el archivo anterior que creamos fue para definir este tipo que vamos a usar.

Definimos también algunos elementos que usaremos:

- contexto, este nos va servir para utilizarlo en el momento que nos soliciten colocarlo.
- *layoutResourceId*, que nos va ayudar a colocar el recurso que deseamos.
- *data[]*, que es un conjunto de datos del tipo “*FrutasVerduras*”.

```

public class FrutasVerdurasAdapter extends
ArrayAdapter<FrutasVerduras>{
    Context context;
    int layoutResourceId;
    FrutasVerduras data[] = null;
}

```

Definimos un constructor en el que recibimos los 3 elementos anteriores que definimos y los asignamos a nuestro elemento.

```

public FrutasVerdurasAdapter(Context context, int
layoutResourceId,
        FrutasVerduras[] data) {
    super(context, layoutResourceId, data);
    // TODO Auto-generated constructor stub
    this.context = context;
    this.layoutResourceId = layoutResourceId;
    this.data = data;
}

```

Tendremos una vista que es la que iremos llenando con los elementos que vamos recibiendo. Utilizamos el *LayoutInflater*, que es lo que se muestra al finalizar todo. El código es el siguiente:

```

public View getView(int position, View convertView, ViewGroup
parent) {

    View convertView;
    FrutasVerdurasHolder holder = null;

    if(row==null){
        LayoutInflater inflater =
((Activity)context).getLayoutInflater();
        row = inflater.inflate(layoutResourceId, parent, false);

        holder = new FrutasVerdurasHolder();
        holder.imagen =
(ImageView)row.findViewById(R.id.imagen);
        holder.texto = (TextView)row.findViewById(R.id.tv);
        row.setTag(holder);
    }else{
        holder=(FrutasVerdurasHolder)row.getTag();
    }

    FrutasVerduras frutasVerduras = data[position];
    holder.texto.setText(frutasVerduras.title);
    holder.imagen.setImageResource(frutasVerduras.icon);

    return row;
}

```

En el código anterior observamos que estamos utilizando un *holder* que viene de *FrutasVerdurasHolder()*, este nos sirve para mantener una referencia a cada uno de los elementos que tengamos que manipular del *layout*, que en este caso son la imagen y el texto.

Lo que buscamos con esto es acceder después de una forma sencilla al elemento. Nos aprovechamos de la propiedad *.getTag().getTag()*.

El *holder* lo podemos considerar como un tema avanzado en Android ya que es un modo de hacer las cosas de mejor forma, si no colocáramos un *holder*, el consumo de recursos de nuestra aplicación sería bastante alto y en aplicaciones con listas largas el rendimiento sería una pesadilla. Tendríamos que ir creando y borrando objetos dependiendo de los elementos que se están mostrando de la lista en la pantalla. Lo que hacemos es crear una clase que guarda las clases la primera vez y luego las recuperamos.

El código es el siguiente:

```
static class FrutasVerdurasHolder{
    ImageView imagen;
    TextView texto;
}
```

Con esto ya tenemos cubiertos todos los puntos que debe cubrir un *adapter*.

Ahora tenemos nuestro archivo principal que es el que se encarga de llenar la lista que vamos a ver y ejecutar la acción en el momento de hacer clic. Su código es (**Archivo MainActivity.java**):

```
package com.example.ejemplolista;

import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
```

```

import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
    ListView lv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FrutasVerduras frutasVerduras_datos[] = new
FrutasVerduras[]{
            new FrutasVerduras(R.drawable.ic_launcher,"Manzana"),
            new FrutasVerduras(R.drawable.ic_launcher,"Pepino"),
            new FrutasVerduras(R.drawable.ic_launcher,"Pera"),
            new FrutasVerduras(R.drawable.ic_launcher,"Naranja"),
            new FrutasVerduras(R.drawable.ic_launcher,"Sandia"),
            new FrutasVerduras(R.drawable.ic_launcher,"Lechuga"),
        };

        FrutasVerdurasAdapter adapter = new
FrutasVerdurasAdapter(this, R.layout.listview_item_row,
            frutasVerduras_datos);

        lv= (ListView) findViewById(R.id.lv);
        ViewgetLayoutInflater().inflate(R.layout.list_header_row, null);
        lv.addHeaderView(header);
        lv.setAdapter(adapter);
        lv.setOnItemClickListener(new OnItemClickListener() {

            public void onItemClick(AdapterView<?>arg0, Viewint
arg2, long arg3){ arg3){
                TextView v = (TextView)arg1.findViewById(R.id.tv);
                Toast.makeText(getApplicationContext(),
v.getText(),Toast.LENGTH_SHORT).show();
            }
        }
    }
}

```

```

        });
    }

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}

```

Creamos la clase, heredamos de *Activity* y declaramos el elemento *ListView* que vamos a utilizar:

```

public class MainActivity extends Activity {
    ListView lv;

```

Después creamos *frutasVerduras_datos[]* que es el contenedor de todos los elementos que queremos colocar en la lista, debemos saber que cuando son muchos los datos que vamos a tener para que no aumente inmensamente nuestro código, se recomienda tomarlo de un XML, *Json* u otro formato de archivo, sería algo similar a lo que haremos en el capítulo 22.

Al no tener muchos elementos podemos hacerlo de forma directa como en este ejemplo, ya que tenemos los datos declaramos el *adapter* que vamos a utilizar y le pasamos los parámetros que nos pide: contexto, el diseño que vamos a utilizar y la fuente de los datos que vamos a colocar:

```

        FrutasVerduras frutasVerduras_datos[] = new
FrutasVerduras[]{
            new FrutasVerduras(R.drawable.ic_launcher,"Manzana"),
            new FrutasVerduras(R.drawable.ic_launcher,"Pepino"),
            new FrutasVerduras(R.drawable.ic_launcher,"Pera"),
            new FrutasVerduras(R.drawable.ic_launcher,"Naranja"),
            new FrutasVerduras(R.drawable.ic_launcher,"Sandia"),
            new FrutasVerduras(R.drawable.ic_launcher,"Lechuga"),
        };

        FrutasVerdurasAdapter adapter = new
FrutasVerdurasAdapter(this, R.layout.listview_item_row,
            frutasVerduras_datos);
    
```

Asignamos el elemento lista que declaramos al *ListView* que tenemos en el archivo XML. Después creamos el *header* que va a ser el encabezado que creamos en un XML y lo asignamos a la lista como su encabezado y para concluir asignamos el *adapter* que creamos anteriormente:

```

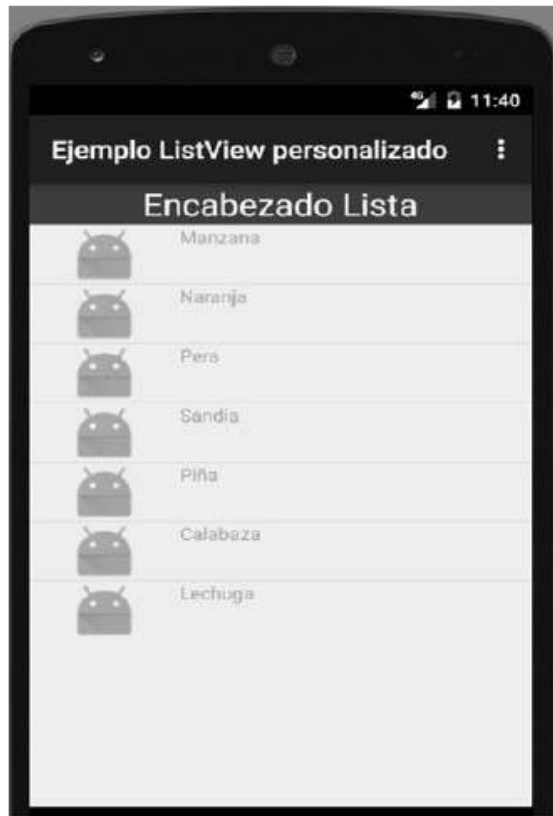
        lv= (ListView) findViewById(R.id.lv);
        ViewgetLayoutInflater().inflate(R.layout.list_header_row, null);
        lv.addHeaderView(header);
        lv.setAdapter(adapter);
    
```

Para poder mostrar el texto de cada ítem dentro de un *Toast*, vamos a implementar el *onItemClickListener* a los elementos de la lista, lo habilitamos para la lista y en el *onItemClickTextView* temporal que nos permite recibir el *TextView* del elemento que fue presionado y pasarlo al *Toast* para imprimir el mensaje:

```

public void onItemClick(AdapterView<?>arg0, Viewint arg2, long
arg3){ arg3){
            TextView v = (TextView)arg1.findViewById(R.id.tv);
            Toast.makeText(getApplicationContext(),
v.getText(),Toast.LENGTH_SHORT).show();
        }
    
```

Esta sería la vista de nuestra aplicación:



Al presionar cualquier elemento nos mostrará el texto que este contenga:



22

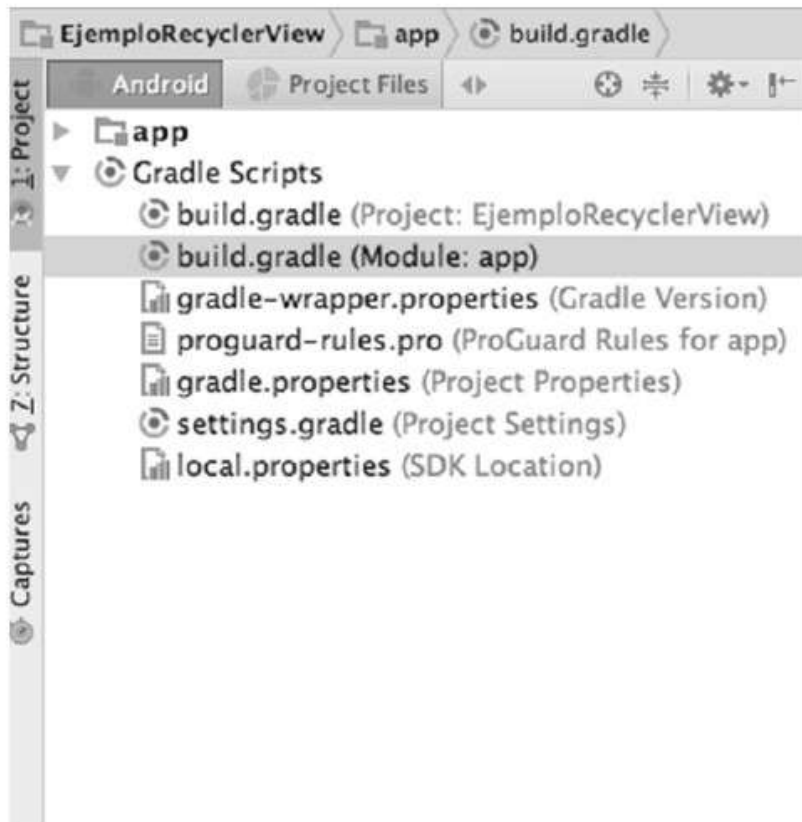
RECYCLERVIEW

El elemento RecyclerView muchas veces viene acompañado de una explicación sencilla pero no del todo correcta, podríamos decir que el RecyclerView es la evolución del ListView, como se mencionó en un principio esto puede ayudar a dejar claro que es un elemento más avanzado y con más opciones que el del capítulo anterior “ListView”, pero en la mayoría de los casos cuando alguien encuentra esta explicación da por hecho que entonces el elemento ListView ya no se usa y ahora todo lo que se hacía con ListView tiene que ser realizado con RecyclerView y esto no es así.

RecyclerView sí es un elemento más avanzado, de hecho la mayoría de las características que tiene ese elemento son gracias a las carencias que presentaba el listview, pero el listview se puede seguir utilizando sin ningún problema, en muchas ocasiones en donde no necesitamos un elemento tan elaborado para mostrar una lista, el ListView sigue siendo una muy buena opción.

Para poder tener RecyclerView en nuestra aplicación lo primero que tenemos que hacer es agregar la dependencia a la misma.

Vamos a ir al archivo build.gradle que maneja las dependencias de nuestro proyecto, lo podemos ver en la imagen:



En el apartado de dependencias tendremos que agregar las siguientes dependencias:

```
dependencies {

    compile fileTree(dir: 'libs', include: ['*.jar'])

    testCompile 'junit:junit:4.12'

    compile 'com.android.support:appcompat-v7:25.0.+'
    compile 'com.android.support:design:25.0.+'
    compile 'com.android.support:recyclerview-v7:25.0.+'

}
```

Como podemos observar en el código tenemos la clásica dependencia:

```
compile 'com.android.support:appcompat-v7:25.0.+'
```

Después tenemos 2 dependencias que se utilizan hoy en día en cualquier aplicación Android:

```
compile 'com.android.support:design:25.0.+'
```

```
compile 'com.android.support:recyclerview-v7:25.0.+'
```

Estas dependencias no faltan en los desarrollos hoy en día, ya que la dependencia de “design” nos sirve para colocar algunos elementos nuevos en una interfaz Android y la dependencia “recyclerview” es para poder implementar este elemento tan usado en Android.

Los números de las versiones van cambiando, lo único que tienes que considerar es en qué versión se encuentra actualmente el elemento, pero el lector no se debe preocupar ya que el mismo Android Studio o la documentación oficial es la que informará de cuál es la versión que se está utilizando en una fecha determinada.

Ahora agregaremos un elemento RecyclerView a nuestro Layout principal:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:orientation="vertical"

    tools:context=".MainActivity">
```

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rv"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
/>
```

```
</LinearLayout>
```

Como elemento principal tenemos un `LinearLayout` que solamente contiene las propiedades que ya hemos descrito anteriormente en este libro como la orientación, el alto, el ancho, etc.:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

El siguiente elemento es el que le da el nombre a este capítulo, RecyclerView:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rv"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
/>
```

Podemos ver que tenemos un id asignado al elemento:

```
android:id="@+id/rv"
```

Después accedemos a la propiedad que nos permite colocar barras para recorrer el elemento y tenemos asignado el valor “vertical”:

```
android:scrollbars="vertical"
```

Como todo elemento visual tenemos asignado un ancho y un alto:

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Ahora vamos a ir al código Java para definir el elemento en nuestra clase MainActivity:

```
public class MainActivity extends AppCompatActivity {
    RecyclerView recyclerView;
```

```

RecyclerView.Adapter adapter;

RecyclerView.LayoutManager layoutManager;

String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento
3", "Elemento 4",
                                "Elemento 5", "Elemento 6", "Elemento 7",
"Elemento 8",
                                "Elemento 9", "Elemento 10", "Elemento 11",
"Eelemento 12",
                                "Elemento 13", "Elemento 14", "Elemento 15",
"Elemento 16"};

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);


    recyclerView = (RecyclerView) findViewById(R.id.rv);

recyclerView.setLayoutManager(new
LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));

    adapter = new RecyclerViewAdapter(Data);

    recyclerView.setAdapter(adapter);

```

```
}
```

```
@Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {
```

```
    // Inflate the menu; this adds items to the action bar if it is present.
```

```
    getMenuInflater().inflate(R.menu.menu_main, menu);
```

```
    return true;
```

```
}
```

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    // Handle action bar item clicks here. The action bar will
```

```
    // automatically handle clicks on the Home/Up button, so long
```

```
    // as you specify a parent activity in AndroidManifest.xml.
```

```
    int id = item.getItemId();
```

```
    //noinspection SimplifiableIfStatement
```

```
    if (id == R.id.action_settings) {
```

```
        return true;
```



```

    }

    return super.onOptionsItemSelected(item);
}
}

```

Comenzamos definiendo los elementos que vamos a usar en nuestra aplicación:

```

public class MainActivity extends AppCompatActivity {

    RecyclerView recyclerView;

    RecyclerView.Adapter adapter;

    RecyclerView.LayoutManager layoutManager;
}

```

El elemento `recyclerview` es el que acabamos de definir, el elemento `adapter` lo vamos a definir y examinar más adelante, lo mismo haremos con el `layoutManager`.

Para llenar nuestro elemento `RecyclerView` vamos a utilizar un arreglo de cadenas, tenemos el siguiente código:

```

String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento 3", "Elemento 4",
    "Elemento 5", "Elemento 6", "Elemento 7", "Elemento 8",
    "Elemento 9", "Elemento 10", "Elemento 11", "Elemento 12",
    "Elemento 13", "Elemento 14", "Elemento 15", "Elemento 16"};

```

A continuación tenemos la definición habitual del método onCreate en donde pasamos el layout que se va a mostrar:

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

Ahora vamos a definir el elemento RecyclerView.

```
recyclerView = (RecyclerView) findViewById(R.id.rv);
```

Debemos de verificar que el id de la línea anterior coincida con el de la definición del XML que hicimos anteriormente, esto es lo mismo que hacemos con cualquier elemento visual, por ejemplo los TextView, los botones, etc.

LayoutManager

El LayoutManager es un elemento que nos ayuda a la distribución de los elementos que vamos a colocar en el recyclerview. En el ListView podemos recordar que cuando queríamos colocar los elementos la opción era lineal de 2 formas, la forma horizontal y la forma vertical, así que nuestros elementos siempre iban colocados uno detrás del otro de forma consecutiva.

Ahora tenemos varias opciones, le podemos decir al layoutManager que queremos los elementos ordenados de forma lineal como ya lo hacíamos anteriormente indicándole que usaremos el acomodo LinearLayoutManager. Otra opción que tenemos es la de un acomodo en tabla, algo muy parecido a un acomodo de filas y columnas como hace la herramienta Excel de Microsoft, este acomodo se llama GridLayoutManager, además tenemos un acomodo considerado como el visualmente más atractivo que es el StaggeredGridLayoutManager, y nos permite tener una tabla con elementos no alineados, podemos decir que es el más complicado de manejar de los tres.

En este caso estando definiendo un elemento de forma vertical, por eso es que definimos un LayoutManager con el método “setLayoutManager” y en la definición

creamos el LinearLayoutManager al que le pasamos 3 parametros, el contexto, el acomodo y un valor booleano, el código es el siguiente:

```
recyclerView.setLayoutManager(new  
LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));
```

Definimos ahora un adapter al que le pasamos el arreglo de las cadenas que definimos anteriormente.

```
adapter = new RecyclerViewAdapter(Data)
```

Este adaptador lo tenemos definido en una clase diferente que la llamamos RecyclerViewAdapter, el nombre lo seleccionamos conforme a nuestras necesidades, por ejemplo si la información que se colocará en el RecyclerView fueran usuarios, se podría pensar en un nombre parecido a UsersAdapter, el nombre en realidad no importa, lo único que es recomendable es que lleve la palabra Adapter en el final para que se puede identificar cuál es su finalidad.

Adapter

Veamos el código del archivo RecyclerViewAdapter:

```
public class RecyclerViewAdapter extends  
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {  
  
    private String[] myData;  
  
    public RecyclerViewAdapter(String[] Data){  
  
        myData = Data;  
  
    }  
  
    @Override  
  
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
```

```
View view = LayoutInflater.from(parent.getContext())
    .inflate(R.layout.item, parent, false);

ViewHolder viewHolder = new ViewHolder(view);

return viewHolder;
}

@Override

public void onBindViewHolder(ViewHolder holder, int position) {
    holder.textView.setText(myData[position]);
}

@Override

public int getItemCount() {
    return myData.length;
}

public static class ViewHolder extends
RecyclerView.ViewHolder{
```

```

        public TextView textView;

        public ViewHolder(View itemView) {
            super(itemView);
            textView = (TextView) itemView.findViewById(R.id.title);
        }
    }
}

```

En este adaptador cuando estamos definiendo la creación vamos a hacer la herencia de “RecyclerView.Adapter” y haciendo referencia a una clase ViewHolder que tenemos definida al final de esta misma clase, el código es este:

```

public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {

```

Continuamos definiendo el elemento que vamos a utilizar dentro de la clase, el nombre es “myData” que es el elemento que tiene el arreglo de cadenas.

```

    private String[] myData;

```

Continuamos con la definición del constructor, en donde indicamos que recibimos el arreglo de cadenas y lo colocamos dentro de la definición que ya hicimos.

```

    public RecyclerViewAdapter(String[] Data){
        myData = Data;
    }

```

Siempre que utilizamos la herencia que ya mencionamos, tendremos 3 métodos que definir:

- onCreateViewHolder
- onBindViewHolder
- getItemCount

Estos 3 métodos son los básicos que debemos de definir en un adaptador de un RecyclerView, el primer método lo que hace es definir una vista que vamos a ir inflando para colocar la interfaz de cada uno de los items, en otras palabras le vamos a indicar cómo se debe de ver cada ítem. Por definición este elemento debe de devolver siempre un elemento que en nuestro caso es un ViewHolder, este es el código:

@Override

```
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
    View view = LayoutInflater.from(parent.getContext())  
        .inflate(R.layout.item, parent, false);  
  
    ViewHolder viewHolder = new ViewHolder(view);  
  
    return viewHolder;  
}
```

El archivo item.xml al que hacemos referencia en el inflate, es un archivo que se llama item que se encuentra en el directorio layout, este contiene el diseño de cada ítem de la lista, a continuación se muestra la línea a la que hacemos referencia:

```
.inflate(R.layout.item, parent, false);
```

Ahora veamos el contenido del archivo item.xml:

```
<?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"

    android:orientation="vertical"

    >

    <TextView

        android:textSize="30sp"

        android:id="@+id/title"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:text="Hola mundo"

        android:padding="10dp"

        />

    <View style="@style/Divider"/>

</LinearLayout>
```

En el código anterior podemos ver que solo tenemos un linearlayout para el acomodo vertical de los elementos y además tenemos un textview que es el contenedor que va a mostrar el texto de cada uno de los ítems de la lista.

Lo más importante es que debe de tener un id el textview y cualquier elemento que vamos a tener en el ítem si queremos acceder a él desde el código del adapter para modificarle atributos o interactuar con él.

El siguiente código que vemos en el archivo del adaptador es el de un onBindViewHolder, en este método lo que hacemos es colocarle el texto al TextView de cada ítem para que muestre el texto que va a ver el usuario, el texto se coloca dependiendo de la posición de los elementos del arreglo de cadenas que recibimos; por eso utilizamos la posición para colocarlos.

```
@Override

public void onBindViewHolder(ViewHolder holder, int position) {

    holder.textView.setText(myData[position]);

}
```

De los tres métodos que mencionamos nos queda por ver el último que tiene una función muy sencilla. Su función es devolver el tamaño de los elementos que tenemos, esto en su mayoría es para ejecutar instrucciones más completas, lo hacemos devolviendo el valor del “largo”, quedaría así:

```
@Override

public int getItemCount() {

    return myData.length;

}
```

Ahora consultamos el último código del archivo del adapter que es la clase que definimos, esta clase la definimos heredando de la clase ViewHolder que ya se encuentra dentro de la clase RecyclerView.


```
public static class ViewHolder extends RecyclerView.ViewHolder{

    public TextView textView;

    public ViewHolder(View itemView) {

        super(itemView);

        textView = (TextView) itemView.findViewById(R.id.title);

    }

}
```

Dentro de la clase lo único que hacemos es colocar los elementos que va a tener el ítem que estamos creando para colocar dentro de la lista que mostraremos, en el caso del ejemplo lo único que vamos a tener es un recyclerview, por eso mismo solamente definimos un textView:

```
public TextView textView;
```

En el constructor podemos ver que solo recibimos una vista, utilizamos el constructor “super”:

```
public ViewHolder(View itemView) {

    super(itemView);
```

Y asignamos el elemento del ítem al elemento que definimos con el nombre de textView:

```
textView = (TextView) itemView.findViewById(R.id.title);
```

Ahora regresamos a la definición del adapter de nuestro archivo Adapter:

```
adapter = new RecyclerViewAdapter(Data)
```

Ya hemos dado un repaso a lo que contiene nuestro adaptador, ya que tenemos creado un adaptador, ahora se lo asignamos al recyclerview y listo.

```
recyclerView.setAdapter(adapter);
```

Si compilamos el ejemplo, deberemos de ver una pantalla como la siguiente imagen:



Comparando RecyclerView y ListView

Si se creó RecyclerView con la idea de que el desarrollador tenga al alcance más y mejores posibilidades para mostrar la información, dentro de esas mejores posibilidades está la de reciclar los elementos que se tienen para mejorar el rendimiento.

En el Listview podemos decir que reciclar los ítems era algo “opcional”, podrías hacerlo o no hacerlo, quedaba al gusto del desarrollador hacerlo a su modo, en el recyclerview podemos decir que se le obliga a hacerlo del modo correcto ya que al tener esos tres métodos definidos por defecto se tiene que hacer “sí o sí”.

Podemos decir que esa sería la principal diferencia entre los elementos, además tenemos todos los aspectos visuales como las animaciones o los efectos que les podemos aplicar a los ítems de manera general o de forma individual a cada uno de ellos.

23

CARDVIEW

El CardView es un elemento nuevo en Android, el elemento RecyclerView del capítulo anterior y el elemento cardview que veremos en este capítulo podemos decir que tienen una relación muy estrecha, normalmente trabajan de forma conjunta.

En este capítulo vamos a colocar un CardView al código y ejemplo del capítulo anterior.

El código de las dependencias va a cambiar, las dependencias serían así:

```
dependencies {  
  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
  
    testCompile 'junit:junit:4.12'  
  
    compile 'com.android.support:appcompat-v7:25.0.+'  
  
    compile 'com.android.support:design:25.0.+'  
  
    compile 'com.android.support:recyclerview-v7:25.0.+'  
  
    compile 'com.android.support:cardview-v7:25.0.+'  
  
}
```

En la última línea del código anterior podemos ver que se agregó una dependencia, la correspondiente al cardview.

Ahora vamos a modificar el archivo item.xml en donde teníamos únicamente el LinearLayout y el TextView, ahora queremos tener un diseño de tarjeta, como lo tienen hoy en día todas las aplicaciones: Facebook, twitter, gmail, etc. El código es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@+id/cardview"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:layout_margin="5dp"

    >

    <LinearLayout

        android:orientation="vertical"

        android:layout_width="match_parent"

        android:layout_height="wrap_content">

        <TextView

            android:textSize="30sp"

            android:id="@+id/title"

            android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"

        android:text="Hola mundo"

        android:padding="10dp"

    />

</LinearLayout>

</android.support.v7.widget.CardView>

```

Podemos observar que tenemos tres elementos: CardView, LinearLayout y TextView.

El primer elemento que es el CardView es ahora el contenedor principal ya que cada ítem ahora es un CardView, esto nos abre el catálogo de oportunidades, por ejemplo podremos colocar efectos o animaciones que nos sirvan para facilitar el uso de la aplicación al usuario.

```

<android.support.v7.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@+id/cardview"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:layout_margin="5dp"

>

```

El CardView tiene dentro un LinearLayout para el acomodo, aunque ahora tenemos solamente un TextView y podríamos pensar que no hace falta, este LinearLayout está ya preparando todo para que en caso de necesitar otro TextView o cualquier otro elemento, este se acomode por debajo del TextView, podemos decir

entonces que sí sobre el elemento, pero también ya estamos dejando todo listo para un posible cambio en la interfaz.

```
<LinearLayout

    android:orientation="vertical"

    android:layout_width="match_parent"

    android:layout_height="wrap_content">
```

Por último tenemos un elemento que es el mismo TextView, de hecho no cambia nada con respecto al capítulo anterior ya que mantiene la misma regla de diseño.

```
<TextView

    android:textSize="30sp"

    android:id="@+id/title"

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:text="Hola mundo"

    android:padding="10dp"

/>
```

En el MainActivity mantenemos todo el código, aquí está el archivo:

```
public class MainActivity extends AppCompatActivity {

    RecyclerView recyclerView;
```

```
RecyclerView.Adapter adapter;

RecyclerView.LayoutManager layoutManager;

String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento 3",
"Elemento 4",
        "Elemento 5", "Elemento 6", "Elemento 7", "Elemento 8",
        "Elemento 9", "Elemento 10", "Elemento 11", "Elemento
12",
        "Elemento 13", "Elemento 14", "Elemento 15", "Elemento
16"};

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    recyclerView = (RecyclerView) findViewById(R.id.rv);

    recyclerView.setLayoutManager(new
        LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));

    adapter = new RecyclerViewAdapter(Data);

    recyclerView.setAdapter(adapter);

}

@Override
```



```
public boolean onCreateOptionsMenu(Menu menu) {

    // Inflate the menu; this adds items to the action bar if it is present.

    getMenuInflater().inflate(R.menu.menu_main, menu);

    return true;

}
```

@Override

```
public boolean onOptionsItemSelected(MenuItem item) {

    // Handle action bar item clicks here. The action bar will

    // automatically handle clicks on the Home/Up button, so long

    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();

    //noinspection SimplifiableIfStatement

    if (id == R.id.action_settings) {

        return true;

    }

    return super.onOptionsItemSelected(item);

}
```

```
}
```

Ahora vamos a ver código del archivo que es nuestro adaptador:

```
public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {

    private String[] myData;

    public RecyclerViewAdapter(String[] Data){
        myData = Data;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item, parent, false);

        ViewHolder viewHolder = new ViewHolder(view);

        return viewHolder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
```

```

        holder.textView.setText(myData[position]);
    }

    @Override
    public int getItemCount() {
        return myData.length;
    }

    public static class ViewHolder extends RecyclerView.ViewHolder{
        public TextView textView;
        public CardView cardView;
        public ViewHolder(View itemView) {
            super(itemView);
            cardView = (CardView)itemView.findViewById(R.id.cardview);
            textView = (TextView) itemView.findViewById(R.id.title);
        }
    }
}

```

El cambio es muy poco ya que el cardview no tiene una alteración en la lógica de nuestra aplicación, pero en caso de querer acceder a este y realizar algún cambio tenemos que asignarle su elemento correspondiente en cada ítem, esto quiere decir que debemos de buscar el id del cardview.

Podemos ver esto en el ViewHolder:

```
public static class ViewHolder extends RecyclerView.ViewHolder{

    public TextView textView;

    public CardView cardView;

    public ViewHolder(View itemView) {

        super(itemView);

        cardView = (CardView)itemView.findViewById(R.id.cardview);

        textView = (TextView) itemView.findViewById(R.id.title);

    }

}
```

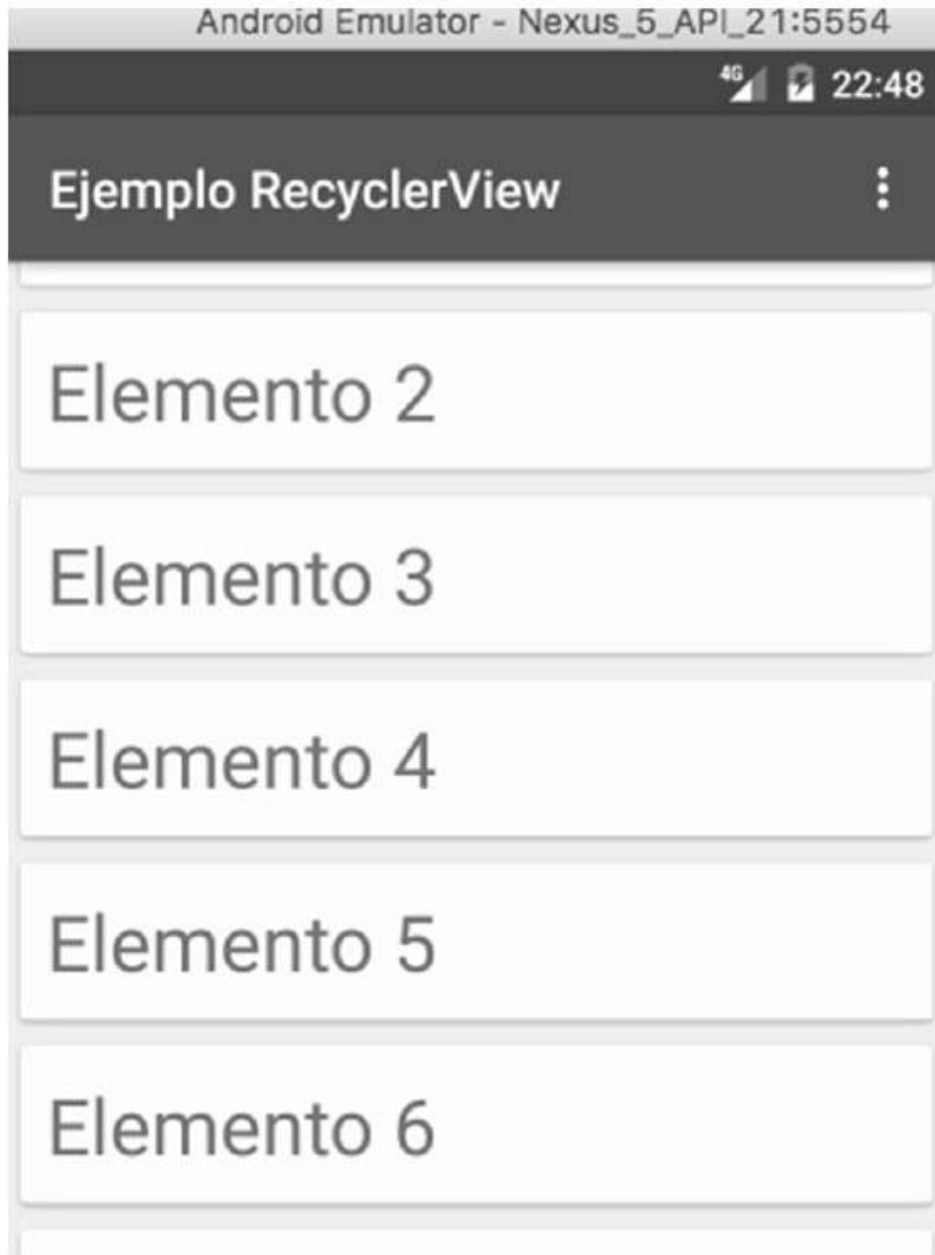
Agregamos la definición:

```
public CardView cardView;
```

Además agregamos el findViewById:

```
cardView = (CardView)itemView.findViewById(R.id.cardview);
```

Con esto ya estamos enlazando nuestros elementos visuales y lógicos. Ahora veamos el cambio de diseño de nuestro recyclerview al agregar el elemento cardview, tendremos algo como lo que se muestra en la siguiente imagen:



Podemos ver que ahora los elementos de la lista son una tarjeta y esto no es solo visualmente, ahora cada uno puede tener comportamientos personalizados, también visualmente pueden ser diferentes y manipularlos por separados.

FLOATINGACTIONBUTTON

El elemento `FloatingActionButton` es un elemento que se agregó dentro de los elementos que forman parte de la pauta “Material Design”, este elemento no solamente es un botón, en realidad tiene objetivos más importantes sobre el uso de las aplicaciones.

En español podremos llamarlo como “botón de acción flotante”, aunque para fines prácticos y para irnos acostumbrando al nombre del elemento lo llamaremos por su nombre en inglés durante este capítulo. En las referencias de la documentación oficial de Android se hace referencia a este elemento como FAB, esta sería la forma corta de mencionarlo.

Este elemento no necesita ser agregado mediante alguna dependencia extra o específica como los elementos `recyclerview` o `cardview`, este ya viene en la dependencia de diseño. La dependencia se llama `Design Support Library`.

*Nota: Para este capítulo utilizaremos el código de los capítulos 22 y 23, los capítulos de los elementos `recyclerview` y `cardview`, respectivamente.

El archivo de las dependencias, `build.gradle` quedaría así:

```
dependencies {  
  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
  
    testCompile 'junit:junit:4.12'
```

```

compile 'com.android.support:appcompat-v7:25.0.1'

compile 'com.android.support:design:25.0.1'

compile 'com.android.support:recyclerview-v7:25.0.+'

compile 'com.android.support:cardview-v7:25.0.+'

}

```

En el momento que se escribe este libro estamos en la revisión 25, pero esta seguramente cambiará con el paso del tiempo, lo importante es que siempre se mantenga actualizada esta parte de las referencias de dependencias.

```

compile 'com.android.support:design:25.0.1'

```

Ahora vamos a ver cómo agregar el elemento visualmente a nuestra actividad, el archivo XML quedaría así:

```

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:showIn="@layout/activity_main"

    tools:context=".MainActivity">

```

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rv"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true"
    android:layout_margin="10dp"
    android:src="@android:drawable/ic_dialog_email" />

</RelativeLayout>
```

Los primeros elementos tanto `LinearLayout` y `RecyclerView` no tienen ninguna alteración, así que en realidad no tenemos que explicar de nuevo qué es lo que hacen.

El último elemento es el que nos interesa, que es el elemento FloatingActionButton:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true"
    android:layout_margin="10dp"
    android:src="@android:drawable/ic_dialog_email" />
```

En la definición del elemento podemos ver que está incluido en la dependencia de diseño y soporte:

```
<android.support.design.widget.FloatingActionButton
```

Continuamos leyendo las líneas y vemos que colocamos un ID que es la referencia al nombre corto que mencionamos en inglés al inicio del capítulo.

```
    android:id="@+id/fab"
```

El elemento tiene una propiedad como la imágenes para colocar la fuente de la imagen (propiedad SRC) que se va a utilizar, en caso de que necesitamos asignarle una imagen podemos utilizar una del directorio drawable, aunque Android ya cuenta con algunas por defecto que podemos utilizar sin mayor problema. Lo que debemos hacer es colocar dentro del atributo src la referencia a esos elementos colocando el “@android”, con eso en el autocompletar de sugerencias vamos a ver muchas opciones.

```
android:src="@android:drawable/ic_dialog_email" />
```

En el ejemplo que estamos creando podemos ver que se optó por una imagen que tiene el diseño de un icono de correo electrónico, así que la imagen parece una carta.

El acomodo del elemento lo hacemos con las siguientes propiedades:

```
android:layout_alignParentRight="true"

android:layout_alignParentBottom="true"

android:layout_margin="10dp"
```

Ahora vamos a definir un elemento FloatingActionButton en el código de la actividad principal, que quedaría así:

```
public class MainActivity extends AppCompatActivity {

    RecyclerView recyclerView;

    FloatingActionButton fab;

    RecyclerView.Adapter adapter;

    RecyclerView.LayoutManager layoutManager;

    String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento
3", "Elemento 4",
                                "Elemento 5", "Elemento 6", "Elemento 7",
"Elemento 8",
                                "Elemento 9", "Elemento 10", "Elemento 11",
"Elemento 12",
                                "Elemento 13", "Elemento 14", "Elemento 15",
"Elemento 16"};
```

@Override

```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    recyclerView = (RecyclerView) findViewById(R.id.rv);
    fab = (FloatingActionButton) findViewById(R.id.fab);

    recyclerView.setLayoutManager(new
        LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));

    adapter = new RecyclerViewAdapter(Data);

    recyclerView.setAdapter(adapter);

    fab.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View v) {

            }

    });

}
```

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {
```

```
// Inflate the menu; this adds items to the action bar if it is present.  
  
getMenuInflater().inflate(R.menu.menu_main, menu);  
  
return true;  
  
}  
  
@Override  
  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    // Handle action bar item clicks here. The action bar will  
  
    // automatically handle clicks on the Home/Up button, so long  
  
    // as you specify a parent activity in AndroidManifest.xml.  
  
    int id = item.getItemId();  
  
  
    //noinspection SimplifiableIfStatement  
  
    if (id == R.id.action_settings) {  
  
        return true;  
  
    }  
  
  
    return super.onOptionsItemSelected(item);  
  
}  
  
}
```

Al inicio podemos observar una línea con la definición del elemento, le ponemos el nombre fab para seguir con la misma pauta del capítulo:

```
FloatingActionButton fab;
```

En las líneas donde definimos los elementos dentro del método onCreate podemos ver la asignación del elemento fab al elemento fab del archivo XML:

```
fab = (FloatingActionButton) findViewById(R.id.fab);
```

Para finalizar la implementación del elemento ahora vamos a definir el onClickListener como si fuera un botón común y corriente, dentro del onClick puedes colocar las instrucciones que te interese ejecutar.

```
fab.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
    }  
  
});
```

Podemos observar que la implementación es bastante sencilla e intuitiva, si sabemos colocar y manipular un botón, un textview o una imagen, no deberíamos de tener problema con la implementación del FloatingActionButton.

Nuestra aplicación se vería de la siguiente forma:



Podemos ver que el botón aparece con el icono del correo y se coloca por defecto en la posición de abajo en la esquina de la derecha, como todos los elementos visuales, en caso de que esto no suceda podemos colocarlo nosotros con algunas propiedades de acomodo como el align, margin, etc.

El color rosa de nuestro FloatingActionButton es el botón que tenemos seleccionado en nuestro archivo de colores como el color Accent. Así que si lo queremos ver con un nuevo color, podemos cambiar el valor del color accent.

El archivo se encuentra dentro del directorio values y se llama colors.xml, este es el código:

```
<resources>

    <color name="colorPrimary">#3F51B5</color>

    <color name="colorPrimaryDark">#303F9F</color>

    <color name="colorAccent">#FF4081</color>

</resources>
```

Recuerda que esta pauta de colores es de Material Design, nos obliga a que seleccionemos 3 colores como paleta de colores mínima, aunque esto lo podemos cambiar nosotros para algunos casos o por necesidad.

La mejor recomendación para trabajar con Material Design es que al menos mantengamos los tres colores: primario, primario oscuro y color para acentuar.

Sentido de usar el FloatingActionButton

El elemento FloatingActionButton tiene un sentido y que es la razón de su existencia. En la mayoría de las implementaciones se puede observar que el botón se encuentra en los lugares de más fácil acceso, esto quiere decir que a primera mano y sin un gran esfuerzo el usuario debe de ser capaz de poder acceder a él y hacer clic.

En el ejemplo del capítulo vemos cómo se coloca en la parte inferior y en el lado de la derecha, esto lo podemos interpretar como un fácil acceso ya que el pulgar de la mano derecha se puede considerar como el más rápido acceso.

Si hacemos un repaso en aplicaciones como Gmail, Facebook, Twitter podemos ver que la acción más recurrente que se piensa, se hace o la más importante es la que acompaña al FloatingActionButton, por ejemplo: en Facebook es hacer una publicación, en Twitter es lanzar un tweet, en Gmail es escribir un correo.

Todas las aplicaciones hoy en día tienen algo claro, cuando uno implementa un FloatingActionButton es para colocar la acción más importante dentro de esta activity. Como desarrolladores debemos de tener claro este mensaje, el FloatingActionButton, además de ser un botón que destaca y de fácil acceso, debe ser un botón con tareas primordiales para usar en la aplicación. Así que podemos resumir en lo siguiente: cuando vemos un botón FloatingActionButton seguramente estamos viendo la acción más importante que el usuario puede ejecutar en esa activity.

25

SNACKBAR

Otro de los elementos que se han vuelto más populares dentro de las últimas versiones de Android en el momento de desarrollar aplicaciones es el Snackbar.

El Snackbar es un elemento que nos sirve como desarrolladores y aplicación para comunicarnos con el usuario y hacerle llegar un mensaje. En la mayoría de las ocasiones se le compara con el Toast, otro elemento con un objetivo similar, el de mostrar un pequeño mensaje por un lapso de tiempo que se encarga de aparecer y desaparecer de la pantalla.

El objetivo primordial del Snackbar es mostrar mensajes cortos, así que ya tenemos algunas pistas de cómo debemos de usarlo y cuándo, si se puede implementar.

El botón aparece en la parte inferior de la pantalla y hace la desaparición hacia la misma parte inferior de la pantalla, su ancho ocupa el ancho de la pantalla por completo.

La dependencia que utilizamos para la implementación de este elemento es la misma implementación que nos facilita el uso del FloatingActionButton, por lo tanto nuestro archivo de dependencias se mantiene idéntico al del capítulo anterior, así sería el código:

```
dependencies {  
  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
}
```



```
testCompile 'junit:junit:4.12'

compile 'com.android.support:appcompat-v7:25.0.1'

compile 'com.android.support:design:25.0.1'

compile 'com.android.support:recyclerview-v7:25.0.+'

compile 'com.android.support:cardview-v7:25.0.+'
```

Al igual que el elemento Toast no tenemos la necesidad de asignarle un espacio en el archivo XML ya que el Snackbar simplemente aparece cuando lo mandamos a llamar y desaparece segundos después de esa aparición, así que podemos saltarnos el archivo XML de nuestra aplicación ya que se mantiene idéntico.

La acción que hará que podamos ver en la pantalla el elemento Snackbar va a ser la que le dio nombre al capítulo anterior del libro que es el “FloatingActionButton”, en el capítulo anterior definimos la acción del onClick aunque no le colocamos código para su ejecución, en este capítulo del libro colocaremos instrucciones que harán que creamos un mensaje del tipo Snackbar.

Así queda nuestro archivo principal:

```
public class MainActivity extends AppCompatActivity {

    RecyclerView recyclerView;

    FloatingActionButton fab;

    RecyclerView.Adapter adapter;

    RecyclerView.LayoutManager layoutManager;

    String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento 3", "Elemento 4",
```

```

        "Elemento 5", "Elemento 6", "Elemento 7",
"Elemento 8",

        "Elemento 9", "Elemento 10", "Elemento 11",
"Elemento 12",

        "Elemento 13", "Elemento 14", "Elemento 15",
"Elemento 16"};

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    recyclerView = (RecyclerView) findViewById(R.id.rv);

    fab = (FloatingActionButton) findViewById(R.id.fab);

    recyclerView.setLayoutManager(new

        LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));

        adapter = new RecyclerViewAdapter(Data);

    recyclerView.setAdapter(adapter);

    fab.setOnClickListener(new View.OnClickListener() {

@Override

        public void onClick(View v) {

            Snackbar.make(v,"Esto es un
Snackbar",Snackbar.LENGTH_LONG).show();

```

```

    }

    });

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);

    return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();

    //noinspection SimplifiableIfStatement

    if (id == R.id.action_settings) {

```

```

        return true;
    }

    return super.onOptionsItemSelected(item);
}
}

```

El método que antes estaba vacío ahora cuenta con la instrucción de crear un Snackbar:

```

fab.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        Snackbar.make(v,"Esto es un
Snackbar",Snackbar.LENGTH_LONG).show();

    }

});

```

Podemos observar en el código anterior que el parecido entre el elemento Snackbar y el Toast es evidente desde la creación del mismo, los dos elementos nos piden un contexto, un texto y una duración. Debemos de recordar siempre colocar el método `.show()` ya que sin esa instrucción el elemento no se va a mostrar.

Así se vería después de presionar el botón:



Acciones en el Snackbar

A continuación vamos a agregar una acción al elemento Snackbar, para eso vamos a seguir con el mismo ejemplo que hemos manejado en los últimos capítulos.

Dentro del archivo XML no tenemos que hacer nada. El archivo que necesitamos modificar es el archivo principal, el archivo de la activity.

Vamos entonces a ver el código del archivo:

```
public class MainActivity extends AppCompatActivity {  
  
    RecyclerView recyclerView;  
  
    FloatingActionButton fab;  
  
    RecyclerView.Adapter adapter;  
  
    RecyclerView.LayoutManager layoutManager;  
  
    String[] Data = new String[]{"Elemento 1", "Elemento 2", "Elemento  
3", "Elemento 4",  
  
                                "Elemento 5", "Elemento 6", "Elemento 7",  
"Elemento 8",  
  
                                "Elemento 9", "Elemento 10", "Elemento 11",  
"Elemento 12",  
  
                                "Elemento 13", "Elemento 14", "Elemento 15",  
"Elemento 16"};  
  
    @Override  
  
    protected void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_main);  
  
  
        recyclerView = (RecyclerView) findViewById(R.id.rv);  
  
        fab = (FloatingActionButton) findViewById(R.id.fab);  
  
        recyclerView.setLayoutManager(new  
  
            LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false));  
    }  
}
```

```

        adapter = new RecyclerViewAdapter(Data);

        recyclerView.setAdapter(adapter);

        fab.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                Snackbar.make(v,"Esto es un
Snackbar",Snackbar.LENGTH_LONG)

                    .setAction("Cerrar",cerrar)

                    .show();

            }

        });

    }

    public View.OnClickListener cerrar = new View.OnClickListener() {

        @Override

        public void onClick(View v) {

            finish();

        }

    };

    @Override

    public boolean onCreateOptionsMenu(Menu menu) {

```

```
// Inflate the menu; this adds items to the action bar if it is present.  
  
getMenuInflater().inflate(R.menu.menu_main, menu);  
  
return true;  
  
}  
  
@Override  
  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    // Handle action bar item clicks here. The action bar will  
  
    // automatically handle clicks on the Home/Up button, so long  
  
    // as you specify a parent activity in AndroidManifest.xml.  
  
    int id = item.getItemId();  
  
  
    //noinspection SimplifiableIfStatement  
  
    if (id == R.id.action_settings) {  
  
        return true;  
  
    }  
  
  
    return super.onOptionsItemSelected(item);  
  
}  
  
}
```


El archivo que ejecuta nuestro Snackbar es el onClick del FloatingActionButton:

```
fab.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        Snackbar.make(v,"Esto es un
Snackbar",Snackbar.LENGTH_LONG)

        .setAction("Cerrar",cerrar)

        .show();

    }

});
```

Dentro del onClick creamos un Snackbar como lo hicimos previamente, sin ningún cambio, le pasamos una vista, el texto y la duración.

Ahora vamos a agregar una instrucción que se llama setAction, con esa instrucción lo que hacemos es agregar una instrucción. Necesitamos pasarle dos parámetros, el primero es el texto que se va a mostrar en el Toast y el segundo es la acción a ejecutar cuando se haga clic sobre el texto que acabamos de indicar:

```
.setAction("Cerrar",cerrar)
```

La pregunta natural a la línea anterior es: ¿Qué es cerrar? Cerrar es un OnClickListener que estamos definiendo en líneas posteriores de nuestro archivo principal:

```
public View.OnClickListener cerrar = new View.OnClickListener() {

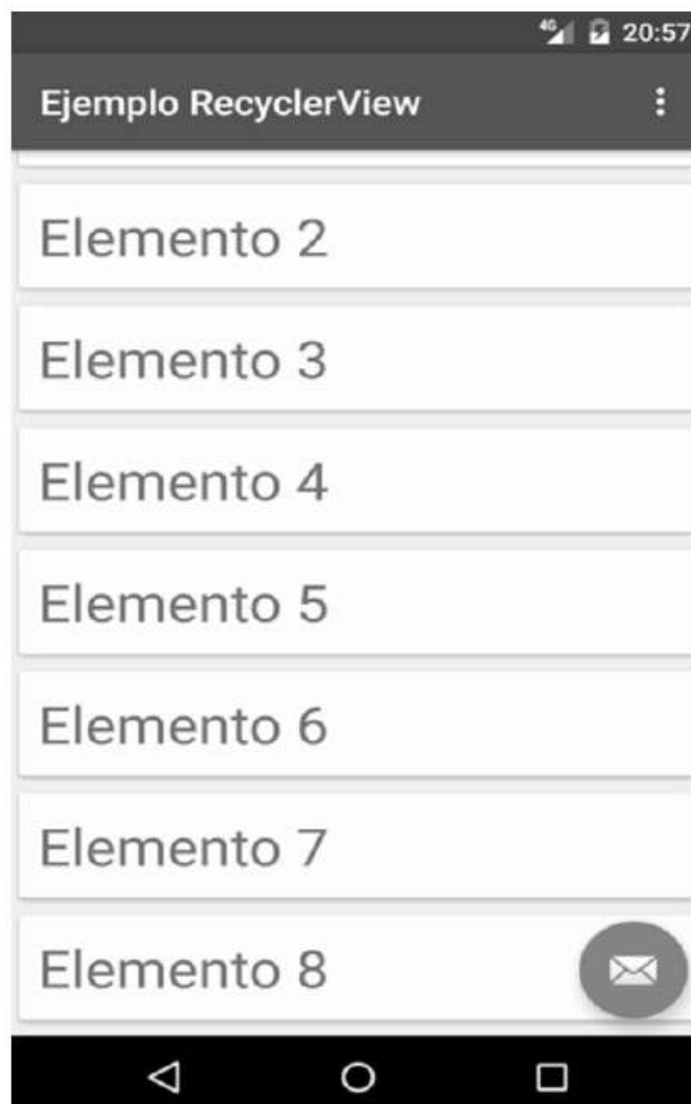
    @Override

    public void onClick(View v) {
```

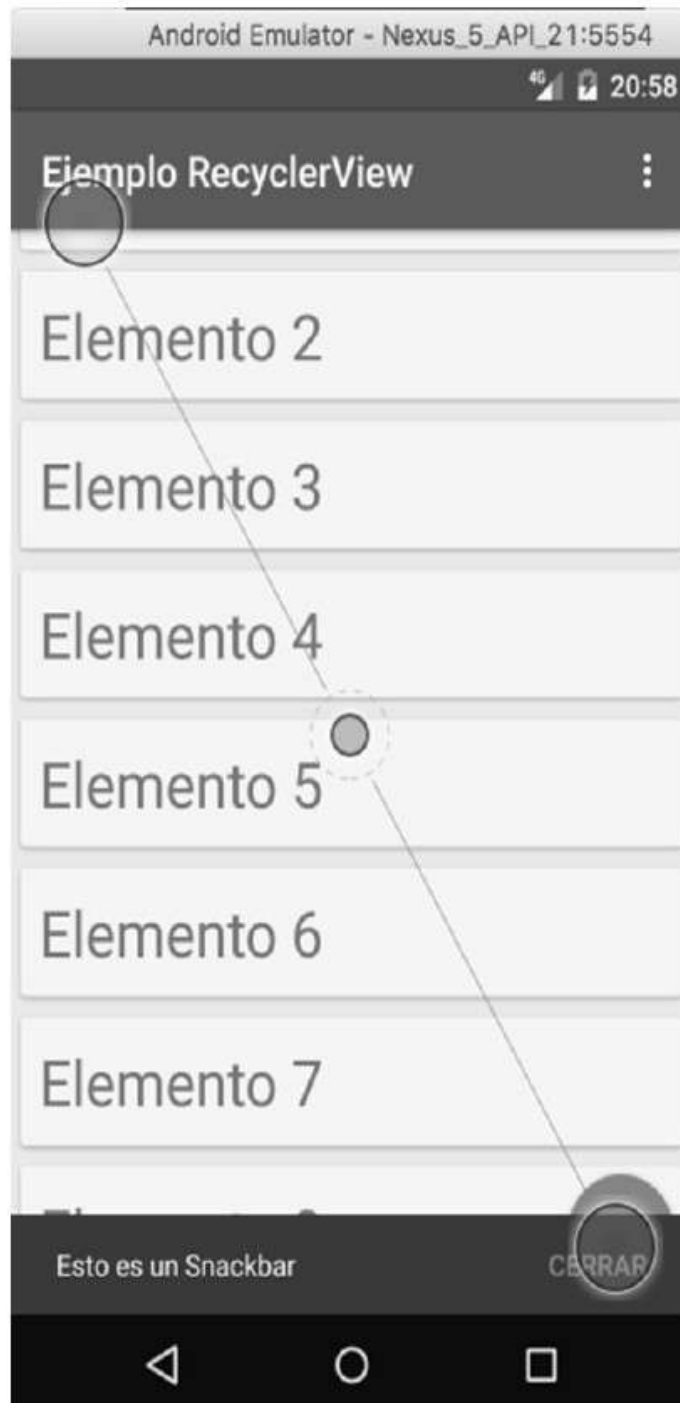
```
        finish();  
    }  
};
```

Cerrar lo estamos definiendo como un `onClickListener` ya que ese es el segundo parámetro que nos piden, por lo tanto podemos colocar en el `onClick` cualquier instrucción o instrucciones que necesitemos, en este caso solamente estamos colocando un `finish()`, esta instrucción nos va a permitir cerrar la activity desde el `SnackBar`.

Al ejecutar la aplicación veríamos algo así:



En el momento de hacer clic tendremos un Snackbar de la siguiente forma:



Podemos observar la opción para cerrar del lado derecho y en ese caso lo que haríamos es hacer clic y la activity se cerraría.

Diferencias entre SnackBar y Toast

Una de las principales diferencias entre estos elementos es que el SnackBar tiene la capacidad de mostrarse de manera indefinida, los dos elementos tienen la medida de duración larga y duración corta. El SnackBar cuenta con una tercera opción que es la duración indefinida, esto haría que el SnackBar se quede en la pantalla por tiempo indefinido.

Otra diferencia importante entre el Toast y el SnackBar es que este último nos permite colocar un texto que nos permite realizar una acción, por ejemplo podremos borrar algo de una lista y si lo deseamos podemos mostrar un texto en un SnackBar que diga que podemos deshacer la acción, en caso de que hagamos clic al texto entonces podremos regresar al estado anterior del elemento.

26 PUBLICAR UNA APLICACIÓN

El proceso de publicar una aplicación es bastante sencillo, pero siempre que es la primera vez debemos de entender algunos conceptos y recibir algunos consejos para hacerlo del mejor modo.

Comenzaremos diciendo que se necesita registrar o darse de alta como desarrollador, lo primero es ir al navegador y abrir Google, escribimos play store console, nos aparecen una serie de enlaces, el primero nos mandará a la consola de la tienda de desarrollo, en esa consola es donde administramos y configuramos las aplicaciones que tenemos publicadas o queremos publicar:



Entramos en la consola de desarrollo y si no estamos registrados como desarrollador debemos de seguir este proceso:

- Iniciar sesión con la cuenta de Google que quieres darte de alta como desarrollador.
- Aceptar los acuerdos y condiciones.
- Pagar el registro por \$25 dólares.
- Llenar la información que te piden.

Esos pasos se ven en las imágenes siguientes:

ANTES DE CONTINUAR...



Consulta y acepta el Acuerdo de distribución para desarrolladores de Google Play.

☐ Acepto las condiciones y quiero asociar el registro de la cuenta con el Acuerdo de distribución para desarrolladores de Google Play.



Consulta los países de distribución en los que puedes vender y distribuir aplicaciones.

Si piensas vender aplicaciones o productos integrados en aplicaciones, comprueba si tienes una cuenta de comerciante en tu país.

 \$25

Asegúrate de tener tu tarjeta de crédito preparada para pagar la cuota de registro (25 USD) en el siguiente paso.

Solamente se necesita una tarjeta de crédito y los 25 dólares, lo demás es como registrarse en una red social.

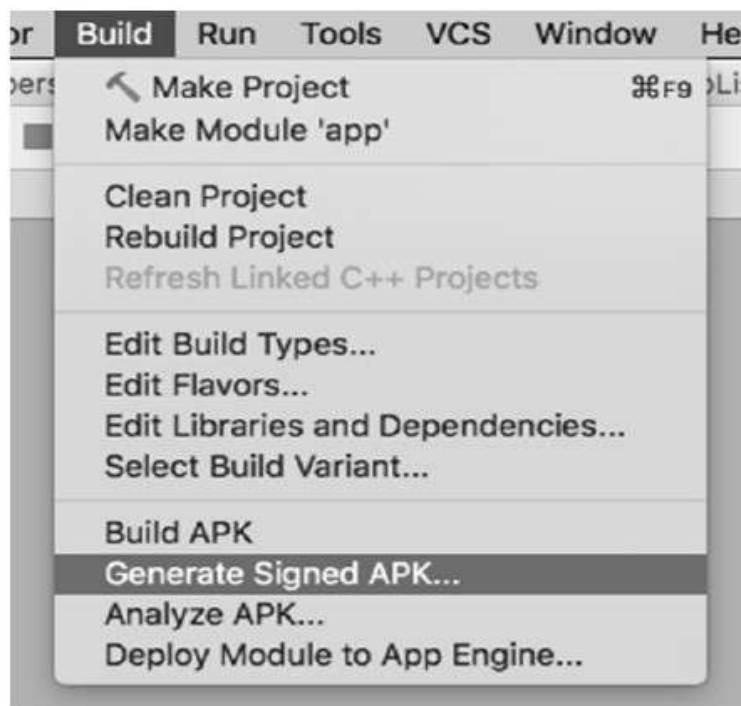
Generar APK

APK es el paquete que se genera desde Android Studio para poder instalar la aplicación en los dispositivos Android. El tamaño de la aplicación es un tema que puede preocupar en algunos casos. APK nos ayuda con este tema ya que es una forma de compresión, por lo tanto, cuando tenemos una aplicación que en código fuente y en los recursos nos da cierto tamaño, este cambiará en el momento de generar el APK ya que comprime el contenido.

Existen dos formas de generar paquetes APK:

- Con firma
- Sin Firma

En este capítulo vamos a ver cómo generar un APK firmado, esto lo hacemos así ya que es el proceso más complejo y si uno quiere poder subir una aplicación a la tienda de aplicaciones esta debe de ir firmada, el otro proceso es mucho más corto y sencillo si lo comparamos.



En la imagen anterior vemos dónde tenemos que ir dentro del menú si nos interesa generar un APK firmado, vamos a ir a la opción Build del menú y después a Generate Signed APK... en esa opción hacemos clic y vemos lo siguiente:



Nos solicita el módulo que queremos, en este caso se selecciona la aplicación y pulsamos en Next y vemos la siguiente ventana:



Por ahora la opción más importante es la de seleccionar una llave nueva o crearla, como es la primera vez que vamos a crear, pulsamos en el botón que dice que la vamos a crear y veremos esto:



Vemos que tenemos diferentes opciones:

- Key store path: la ruta donde colocaremos la llave
- Password: una contraseña
- Confirm: confirmar contraseña
- Alias: un alias para la llave
- Password: contraseña
- Confirm: confirmar contraseña
- Validity: años de validez de la llave, puede ser 99 años
- First and Last Name: nombre
- Organizational Unit: Equipo dentro de la organización
- Organization: Organización
- City: Ciudad
- State or providence: Estado
- Country code: Código del país. Por ejemplo MX en México

No es necesario llenar todos los campos, solo los primeros 8.

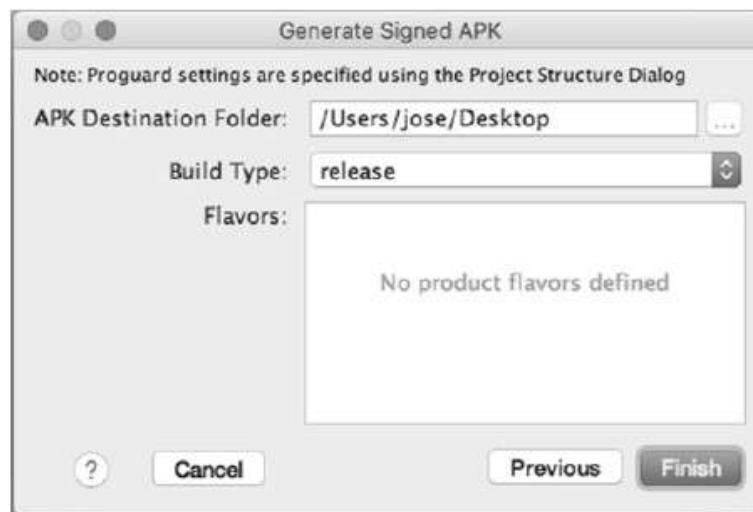


Al pulsar en “OK” veremos la ventana anterior cargada de los datos, es muy importante que esa llave se guarde en un lugar donde se recuerde y se tenga de nuevo acceso, en el caso de que nuestra aplicación ya esté publicada y queramos actualizarla, la tienda de aplicaciones nos va a solicitar la llave.

En caso de no tener la llave no podremos bajo ninguna circunstancia actualizar la aplicación, tendremos que acabar creando otra aplicación en la tienda, esto es algo importante si es que le queremos dar seguimiento al proyecto que queremos subir a la tienda de aplicaciones:



Al pulsar en “Next” nos aparece una ventana en donde nos pregunta sobre el destino del APK:



Seleccionamos el directorio en donde queremos colocar el APK firmado. Al pulsar en Finish tenemos finalmente el APK después de un tiempo en el directorio que indicamos, esto puede tardar un tiempo, es cuestión de tener paciencia. Felicidades ya tenemos el APK de nuestra aplicación.

Subir APK a la Play Store

Desde la misma consola podemos pulsar en el botón para registrar nuestra aplicación o desde el navegador escribimos lo siguiente:

<https://play.google.com/apps/publish/>

Ahora vamos al botón que dice que podemos subir nuestra aplicación e iniciamos el proceso para subir la aplicación.

Llenaremos el primer formulario:

AÑADIR NUEVA APLICACIÓN

Idioma predeterminado *

Español (Latinoamérica) – es-419

Nombre *

Aplicacion X

12 de 30 caracteres

¿Cómo te gustaría empezar?

Subir APK

Preparar ficha de Play Store

Cancelar

Colocamos el idioma predeterminado para nuestra aplicación y además el nombre, que no debe pasar de 30 caracteres. Después seleccionamos subir el APK y lo buscamos en nuestra computadora y lo subimos, es como adjuntar un archivo a un correo.

SUBIR NUEVO APK DE PRODUCCIÓN

Subiendo app-release.apk
procesando...

Cancelar

Ya que lo subimos ahora vamos a una ventana en donde comienza el proceso de llenar la información de la aplicación.

Vamos a tener que ir pasando cada una de las pestañas y opciones que están a la izquierda para dejar todo listo y poder publicar la aplicación.



Por ahora solo hemos cumplido con subir con el APK como lo muestra la imagen interior, así que vamos a ir seleccionando las opciones en ese mismo orden.

Ficha Play Store

La ficha de información del producto nos indica en cada caso y en cada pestaña del idioma muy bien qué es lo que tenemos que introducir:

- Título de la aplicación: máximo 30 caracteres.
- Descripción breve: máximo 80 caracteres.
- Descripción completa: máximo 4000 caracteres.

Encontraremos algo como la siguiente imagen:

INFORMACIÓN DEL PRODUCTO

Inglés (Estados Unidos) – en-US Idiomas (2) ▼

Título*
Inglés (Estados Unidos) – en-US Maya - l
26 de 30

Descripción breve*
Inglés (Estados Unidos) – en-US Know m
45 de 80

Descripción completa*
Inglés (Estados Unidos) – en-US This app word or bar, the alphabe

Más abajo nos solicita el material gráfico para cada una de las resoluciones para las que está preparada nuestra aplicación: teléfono, 7 pulgadas, 10 pulgadas y TV si es el caso:



A continuación, nos solicita:

- Icono de alta resolución que tiene que ser de 512x512 px.
- Imagen destacada de 1024x500px.
- Imagen promocional 180x120px.
- Banner de TVpx.
- Vídeo promocional, enlace de este a YouTube.

Icono de alta resolución *

Inglés (Estados Unidos) – en-US (predeterminado)

512 x 512

Archivo PNG de 32 bits (alfa)



Imagen destacada

Inglés (Estados Unidos) – en-US (predeterminado)

1024 x 500

Archivo JPG o PNG



Banner de TV

Inglés (Estados Unidos) – en-US (predeterminado)

320 an x 180 al

Archivo JPG o PNG de 24 bits (no alfa)

Finalmente colocamos la categoría del producto, además de los datos de contacto como un sitio web, correo electrónico y teléfono.

Luego agregamos una política de privacidad si es necesario apuntándose a una web con esa información, solo colocando la *URL*.

Ahora tendremos que dar una categoría y llenar algunos datos para poder clasificar mejor la aplicación que estamos subiendo.

CATEGORIZACIÓN

Tipo de aplicación *

Categoría *

Clasificación de contenido *

Nueva clasificación de contenido *

DATOS DE CONTACTO

Sitio web

Correo electrónico *

Teléfono

Los datos son:

- Tipos de aplicación
- Categoría
- Clasificación de contenido

Después necesitamos poner los datos de contacto.

- Sitio Web
- Correo electrónico
- Teléfono
- Política de privacidad

Con eso concluimos esta parte.

Precio y distribución

En el caso de que nuestra aplicación sea gratuita solo tenemos que asignarle que es gratuita en la pestaña de “Precio y distribución”.

Vamos a seleccionar país por país en los que queremos que nuestra aplicación esté disponible, lo recomendable es seleccionar los países en donde se hablan los idiomas que tiene nuestra aplicación.

En el caso de desear cobrar por la aplicación el paso es mucho más complicado, tenemos que crear una cuenta de comerciante y este procedimiento varía un poco para cada país, comenzando por el caso de que en unos países no se permite y en otros sí.

Solo tenemos que seguir los pasos que nos va a ir proporcionando Google para cada país, en realidad no los explicamos en detalle ya que para cada país de habla hispana tendríamos que explicar las excepciones y características.

En el caso de que nuestra aplicación sea gratuita solo tenemos que ir marcando las opciones que nos coloca y dejar las que necesitamos marcadas.

Productos de compra en aplicaciones

En este apartado se ven los productos que vamos a vender dentro de nuestra aplicación, esta opción lo que hace es registrar los elementos que estaremos vendiendo en nuestra aplicación para poder cobrarlos al usuario, como en el libro no se vio cómo crear contenido para su venta esta pestaña la pasamos sin problemas.

Finalizar publicación

Si ya subimos la mínima información de la ficha de la Play Store, además ya colocamos el precio y aceptamos las políticas de distribución, en la parte superior izquierda encontraremos un botón al que tendremos que hacer clic para publicar la aplicación, este botón es el que aparece desde que subimos el APK o entramos a llenar la ficha de descripción e información, pero solamente se habilita hasta que tenemos el mínimo:



Para publicar la aplicación, debes rellenar los campos marcados con *.

Finalmente, en un lapso de máximo 48 horas encontraremos la aplicación en la Play Store, puede ser que desde 4 horas después de hacer clic en la aplicación en la opción *publicada* la encontremos en la tienda.

Recomendaciones al publicar una aplicación

Estas recomendaciones las tengo publicadas en un artículo en Internet y en un libro anterior de Android, aún son vigentes hasta la fecha de hoy.

- Tener un sitio web de la aplicación, sea de pago o gratis el tener un sitio web de nuestra aplicación, con más información o con los datos de contacto genera confianza en la marca y en nuestro trabajo, además de

que también se pueden dar a conocer otras aplicaciones o hasta trabajos para otras plataformas.

- Revisar comentarios en la Play Store, actualmente ya se pueden contestar comentarios que los usuarios realizan en la Play Store, es importante atenderlos y escucharlos.
- Crear las imágenes específicas para cada tamaño, ya que nos permite enseñarle al usuario de forma real cómo se va a ver la aplicación cuando este la instale.
- Actualizar la aplicación constantemente, cuando una aplicación se descarga en muchos casos se abandona, esto ocasiona el malestar del usuario y más si es una aplicación de pago, si mantenemos constantes mejoras y resolvemos problemas que se presenten, ganamos un usuario feliz.
- Mantener controladas las estadísticas, estas nos arrojan bastante información y posibles oportunidades de mercado si sabemos qué es lo que sucede, por ejemplo, qué dispositivos y marcas son los que más usan los usuarios con nuestra aplicación, entre otros.
- Acercarnos a blogs que generen reseñas de aplicaciones, muchos usuarios toman como puntos de referencias estos, para ahorrar tiempo y para estar cerca de las tendencias, que obtengamos alguna reseña de estos blogs puede acelerar el número de usuarios que descarguen la aplicación.
- Colocar nuestros datos de contacto de primera mano en la tienda de aplicaciones, muchas veces los usuarios buscan respuestas sencillas y concretas, si colocamos un mail o medio de contacto, ayuda a que el usuario sienta que no está solo en el momento de descargar y tener dudas de nuestra aplicación.
- Solo dejar nuestra aplicación descargable para los idiomas que la hicimos, si la dejamos para más lenguajes de los que soportamos, generaríamos malas valoraciones que a largo plazo se vuelven negativas.
- Tener siempre en mente los tamaños de pantallas, densidades de pantalla y modos de vista, vertical y horizontal, ya que estos hacen más atractiva la usabilidad para el usuario.

ÍNDICE ANALÍTICO

A

Activity, 45, 173
Adapter, 260
APK, 305
AppCompatActivity, 45
Application Name, 26
AVD, 18, 21, 22

B

Button, 76

C

CardView, 269
Clase, 150
Company Domain, 26

D

Densidades, 103
distribución, 96
drawable, 39

E

Eclipse, 7, 8, 221, 240
Emulador, 17, 19
Escala de imágenes, 145

F

FloatingActionButton, 279
Fragmentos, 211
Fragments, 45, 58
FrameLayout, 58
Framework, 5, 6

G

Glass, 28
Gradle, 32, 41, 42

H

Herencia, 150, 153

I

ImageView, 140
Intent, 176

L

Landscape, 115
layout, 39, 46, 47
LayoutManager, 259
Layouts, 51
Libraries, 5
LinearLayout, 53, 54, 55, 56, 67, 68

M

Margin, 88
MediaPlayer, 195
minSdkVersion, 44
mipmap, 40

N

Netbeans, 7

O

Objeto, 150, 152
Orientación de Pantalla, 111

P

Píxeles, 107, 109
Pulgada, 108
Puntos, 109

R

raw, 196
RecyclerView, 251

S

Sensores, 203
SnackBar, 289
String, 133

T

targetSdkVersion, 44
Tamaños de pantalla, 104
TextView, 67
Toast, 163
TV, 20, 28, 312

W

Wear, 28
WebView, 189, 190, 191, 192