ANDROID STUDIO

Aprende a desarrollar aplicaciones



José Dimas Luján Castillo

Alfaomega

libros R

Android Studio

Aprende a desarrollar aplicaciones

José Dimas Luján Castillo

Alfaomega

libros

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

© Alfaomega - RC Libros

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

© Alfaomega - RC Libros

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

© Alfaomega - RC Libros

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

© Alfaomega - RC Libros

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, ademas 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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

2 ¿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

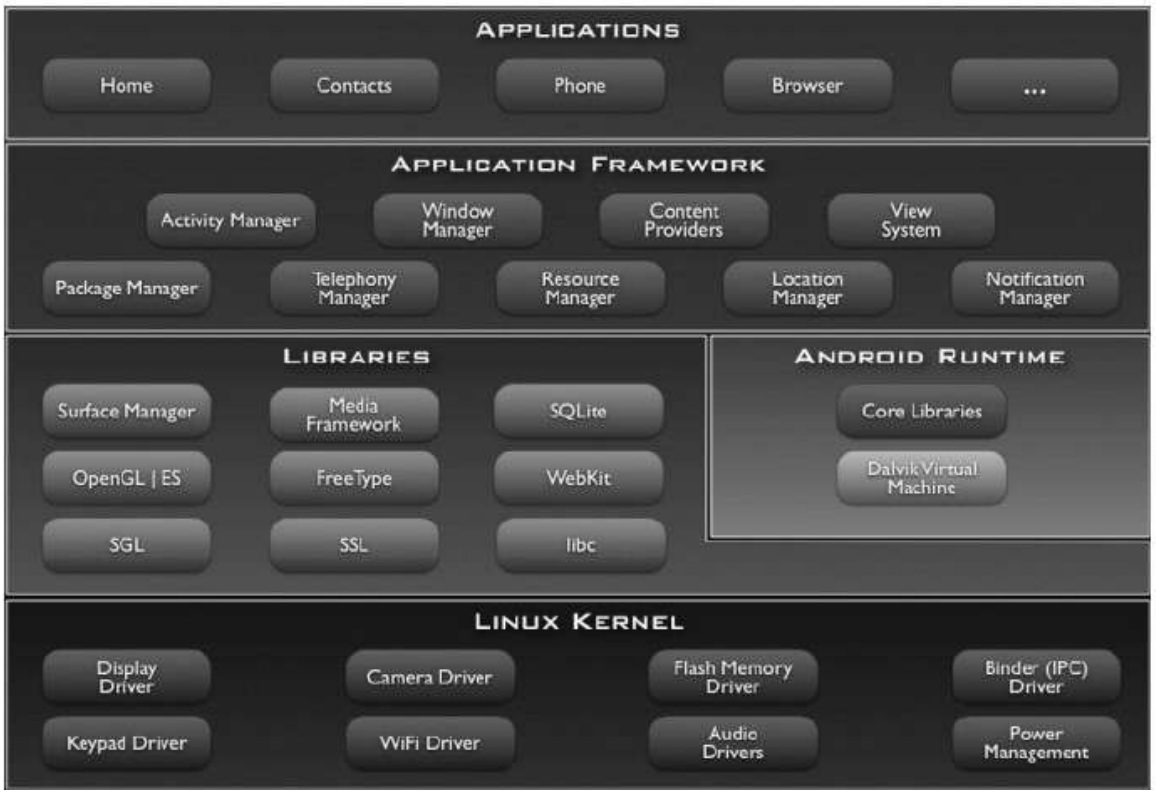
Los nombres de las versiones de Android siempre están en orden alfabético, es decir, que la primera versión se inició con la letra "A", la segunda con la "B" y así

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:



© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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

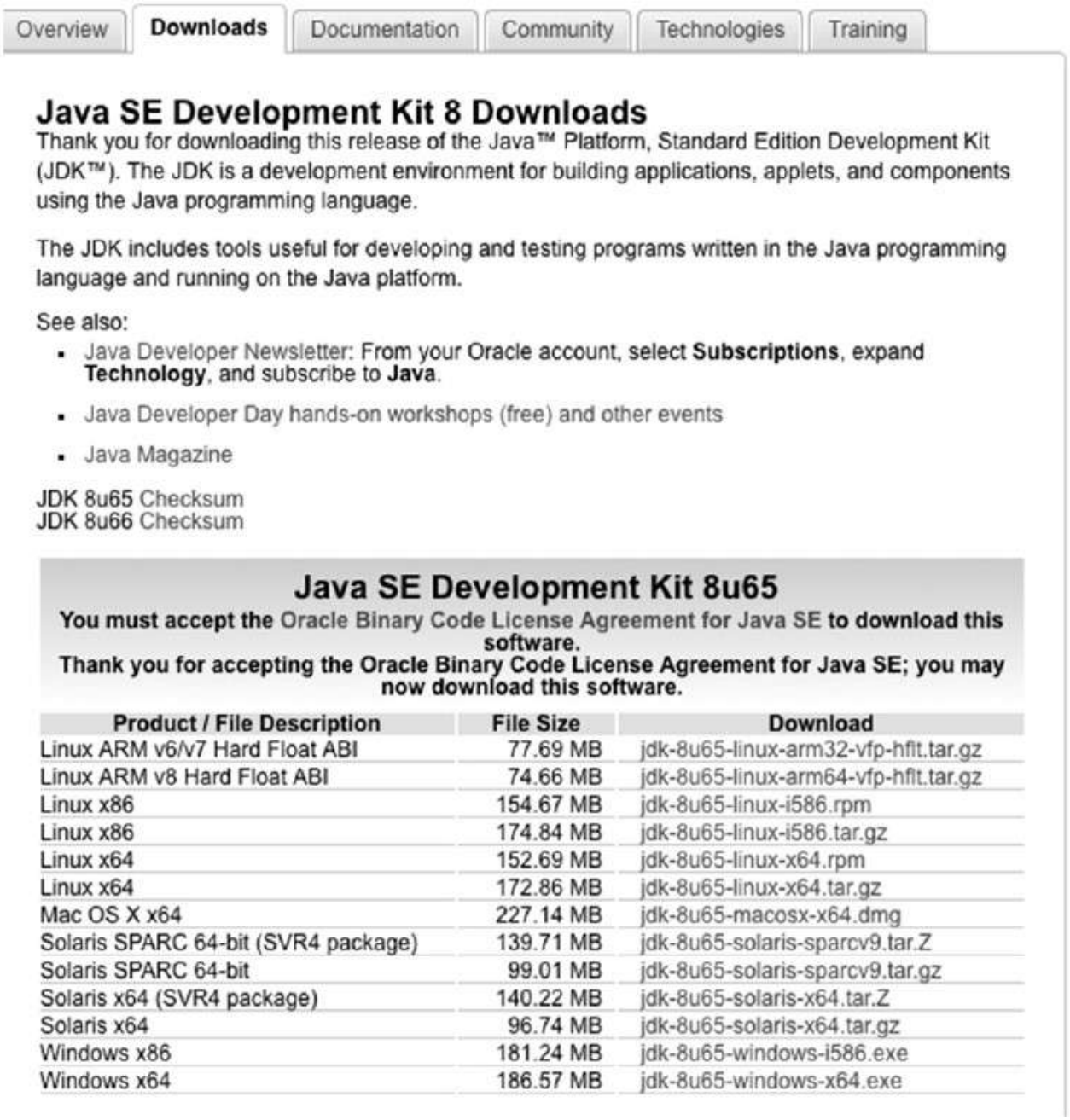
© Alfaomega - RC Libros

Saldrá una imagen como la siguiente:



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.

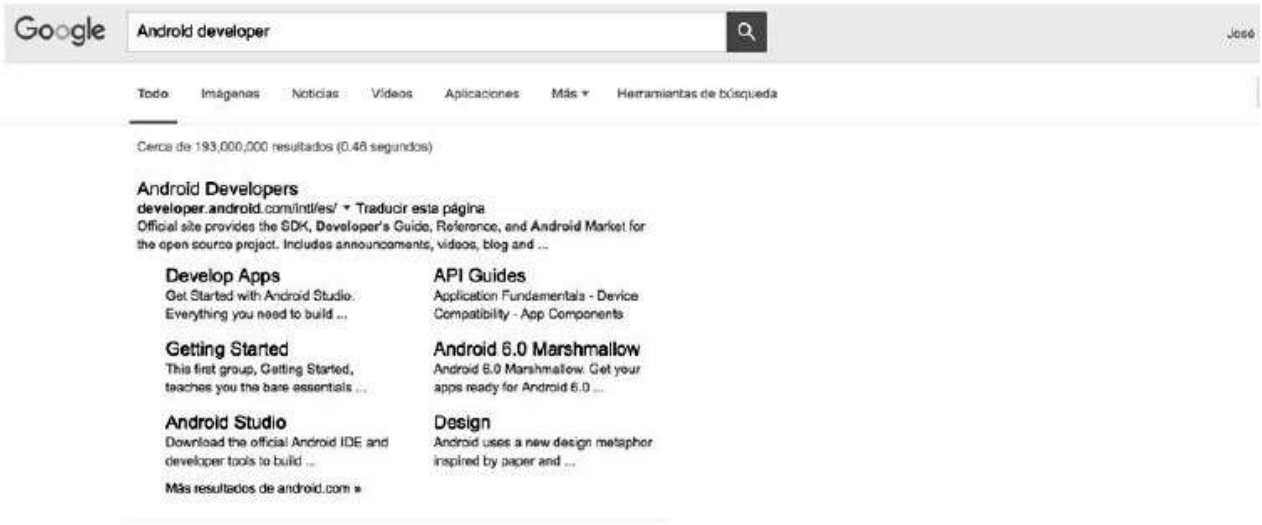
Android Studio

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

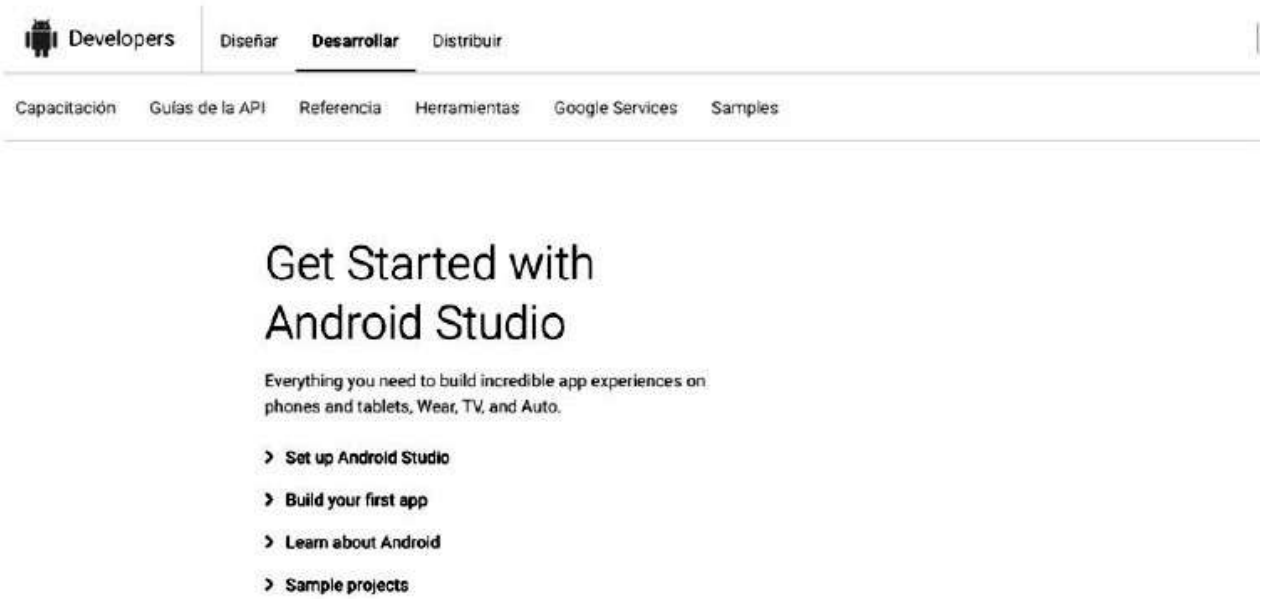
© Alfaomega - RC Libros

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

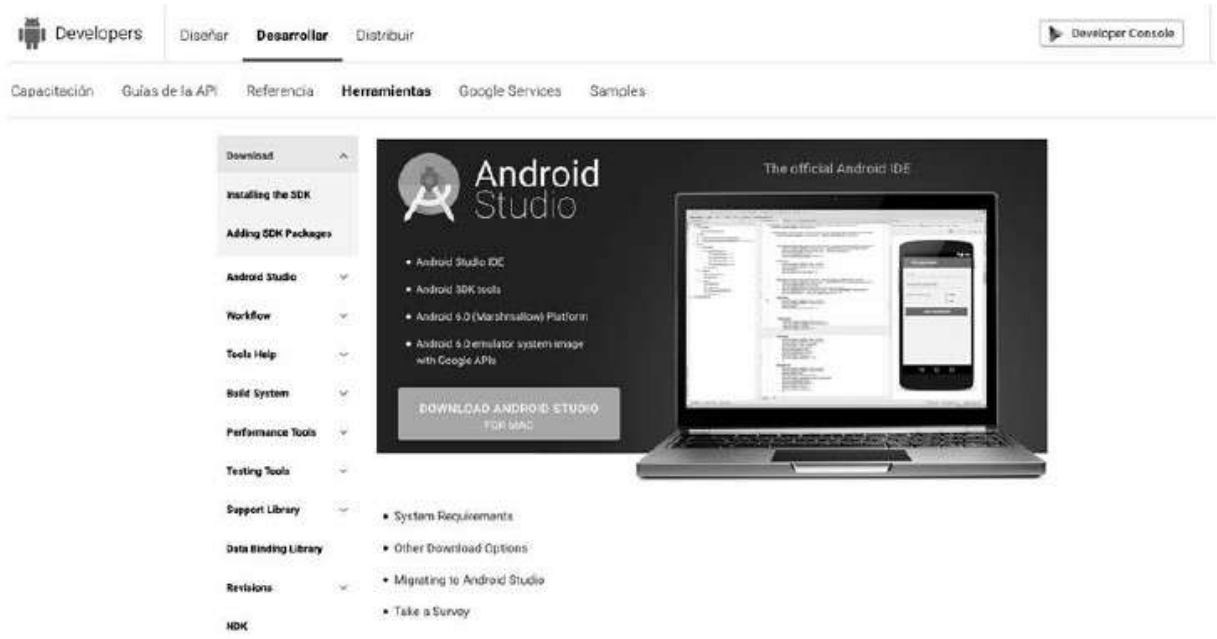
http://developer.android.com



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

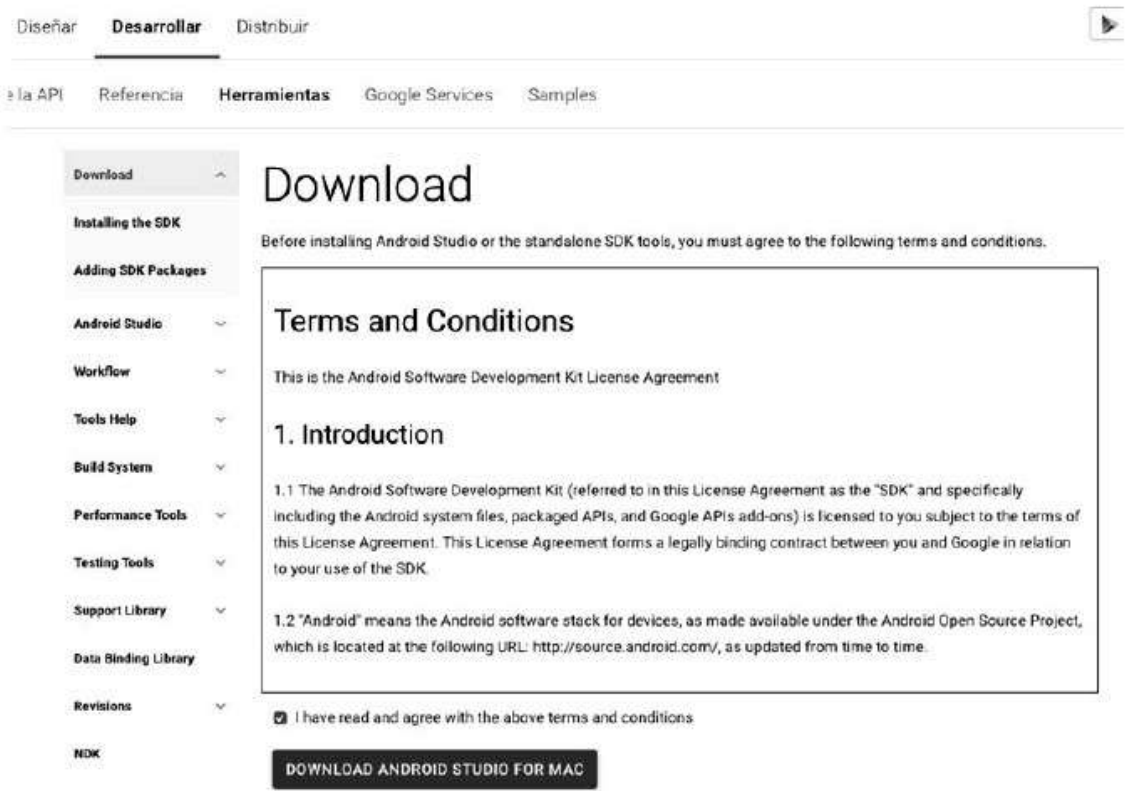


Ahora tenemos un pequeño menú con opciones en la parte central de la pantalla, vamos a la primera opción “Set up Android Studio” y vemos una ventana como la siguiente imagen:



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.

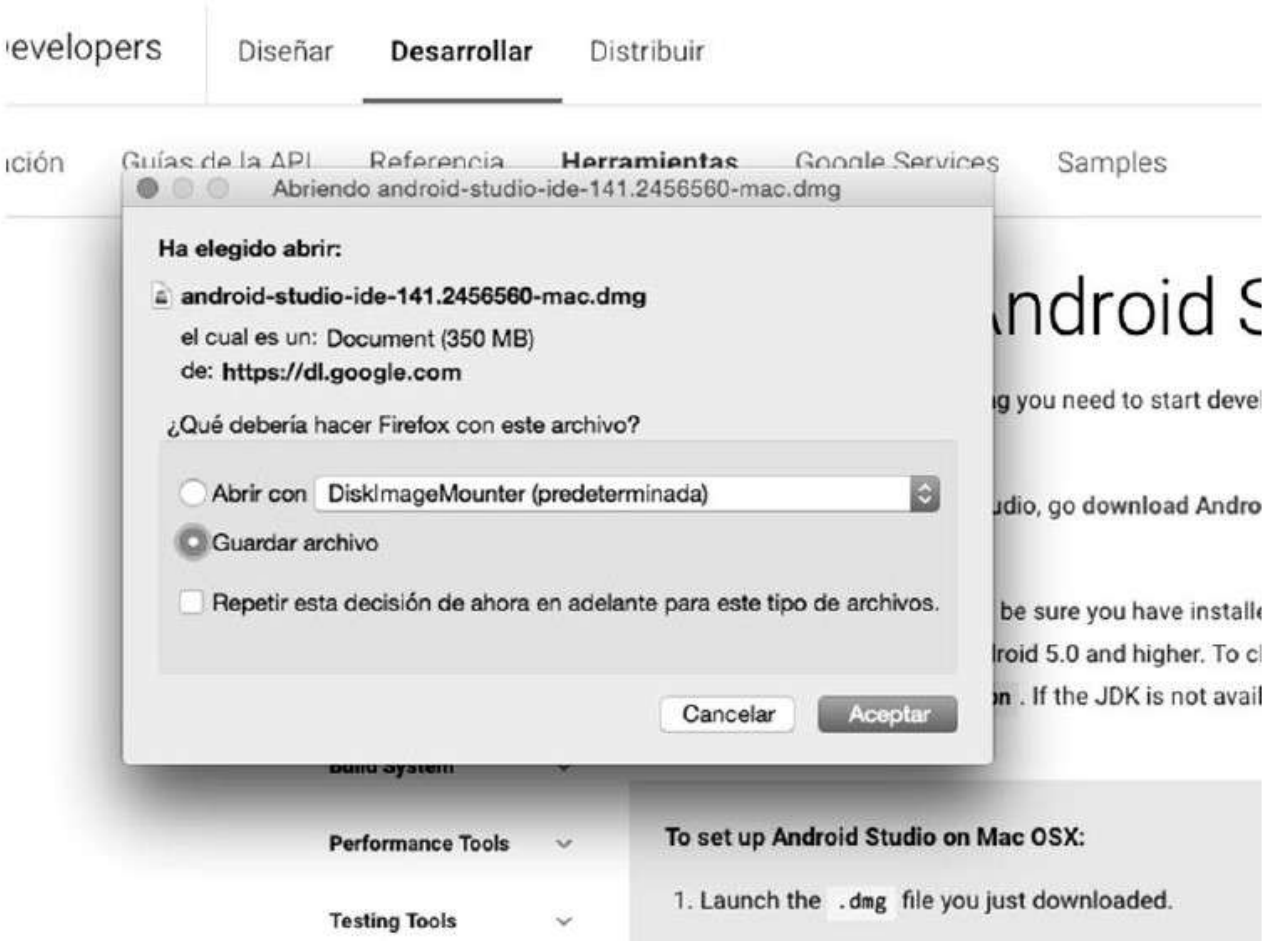
Al hacer clic en el botón veremos una pantalla como la siguiente:



© Alfaomega - RC Libros

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

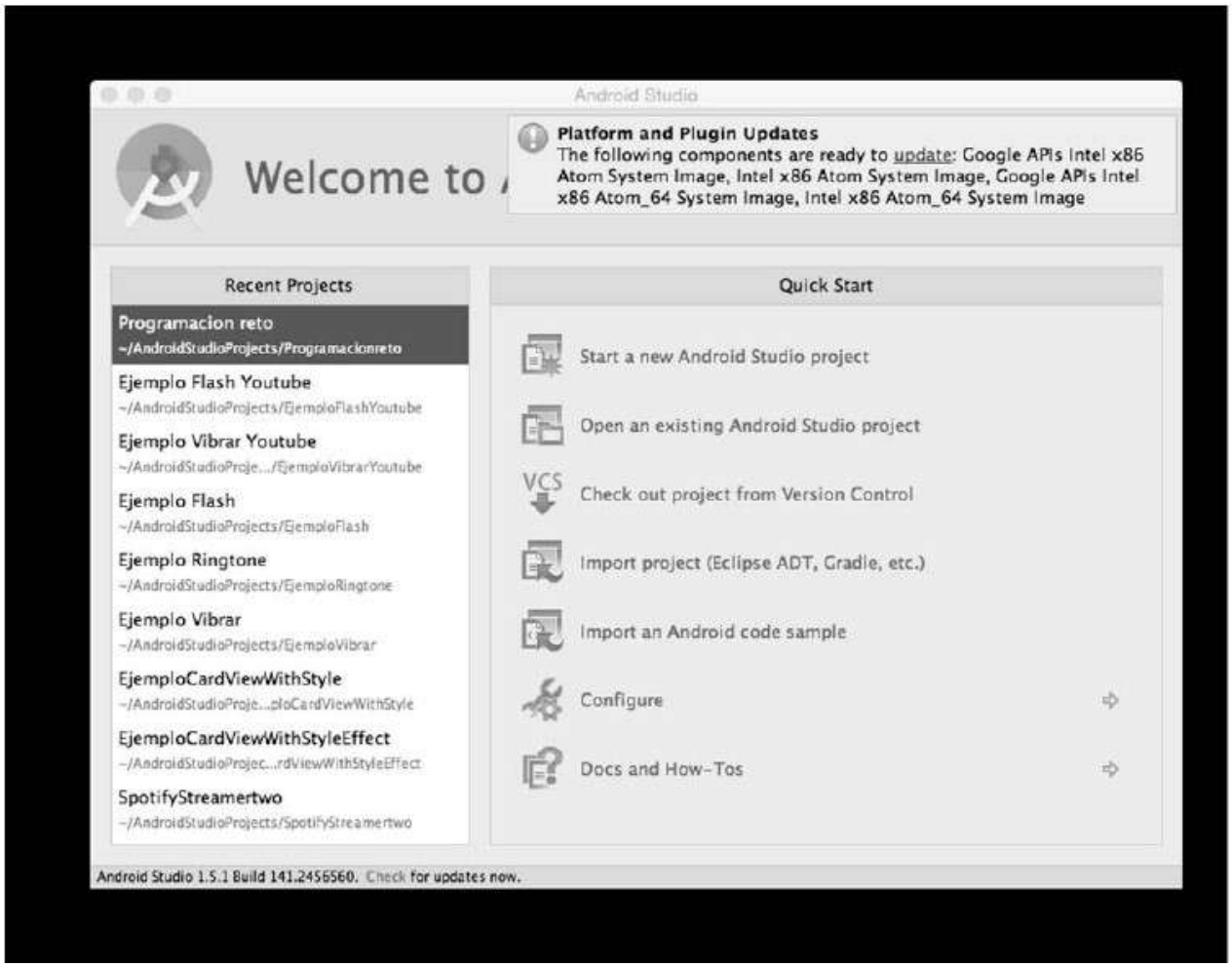
Demos un repaso rápido a las versiones de Android, algo que debemos observar es que como desarrollador nos importa mucho más el número de la API, ya que ahí es donde reside la mayoría de las cosas que utilizamos como desarrolladores.

- 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:

© Alfaomega - RC Libros



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

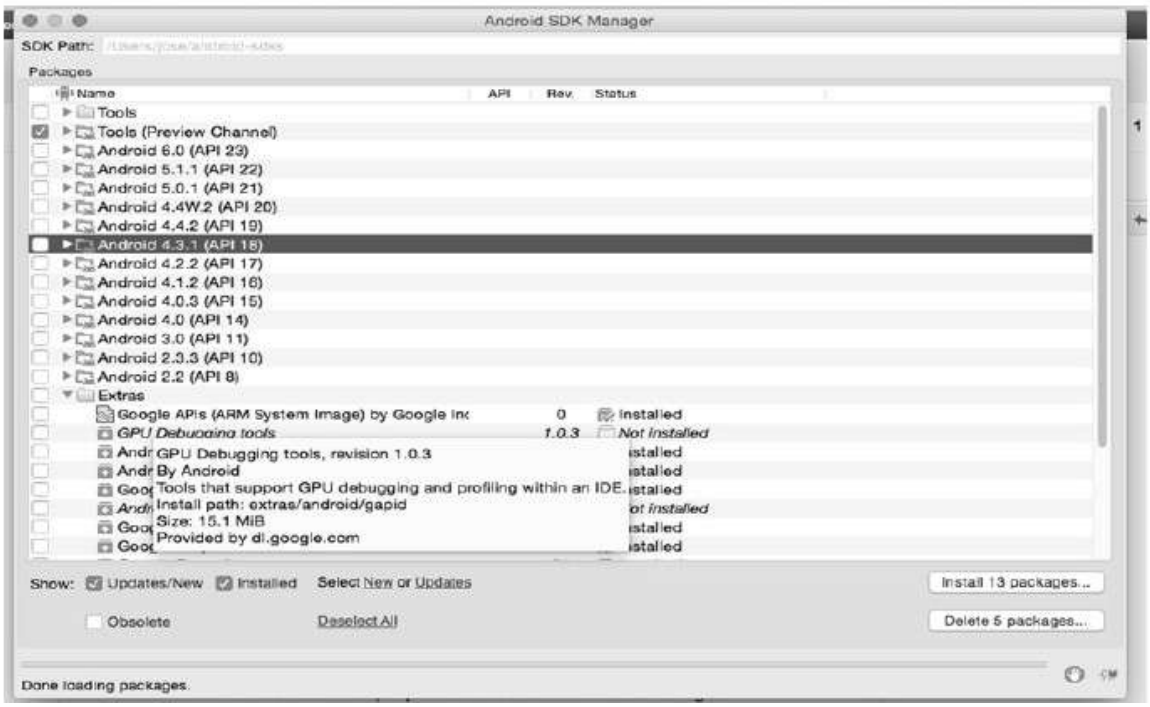
El siguiente paso y muy importante es obtener las versiones de Android para las que queremos programar. No es necesario descargar todas las versiones de Android existentes, pero muchos desarrolladores acostumbran a hacerlo, aunque el proceso de descargar ocupará algunos GB, es recomendable hacerlo un 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

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

Emulador

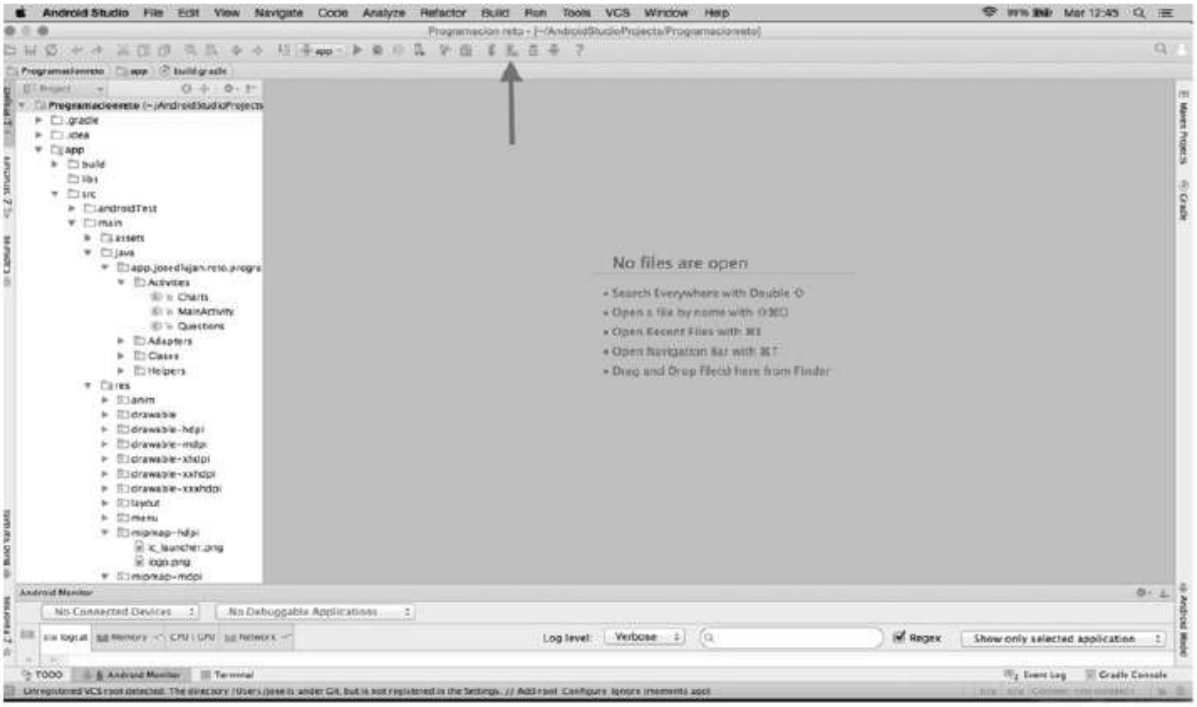
Uno de los beneficios que podemos encontrar al descargar las herramientas oficiales de desarrollo, es el poder probar nuestra aplicación en el emulador. El emulador nos ayuda a no tener que comprar la gran cantidad de dispositivos en los queremos probar nuestra aplicación, sabemos de la gran fragmentación 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.

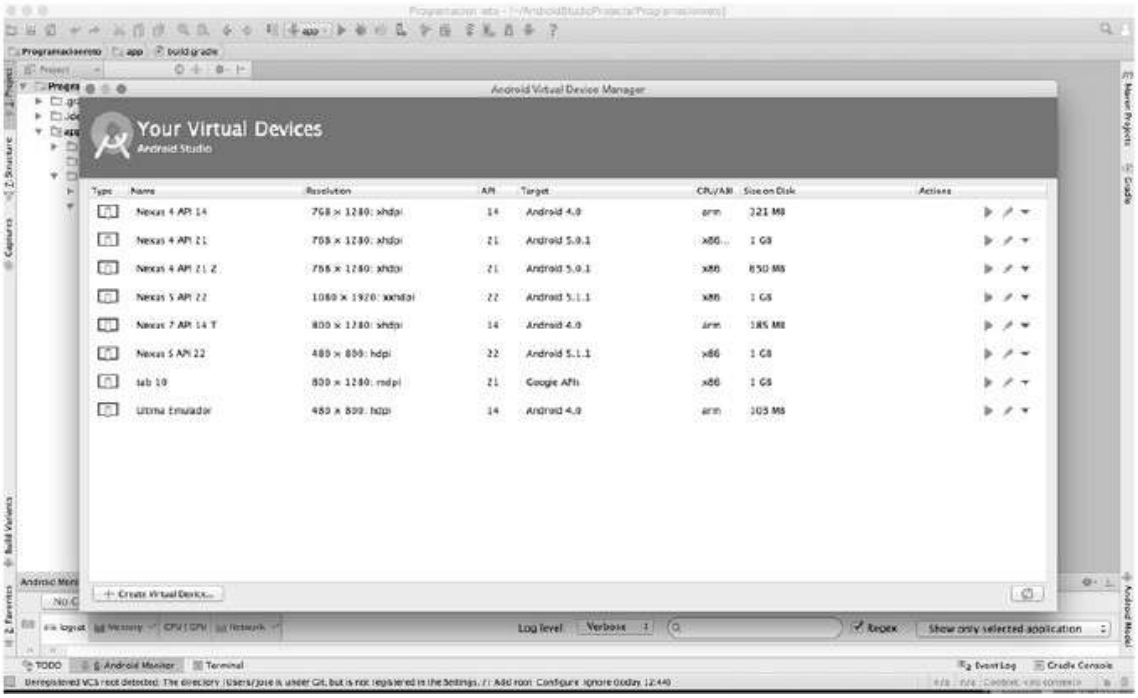
Las recomendaciones anteriores en los puntos son en relación con mi experiencia propia y además recordemos que los sistemas operativos sufren actualizaciones, también Android Studio las tiene, así que en cualquier momento se pueden mejorar o arreglar los problemas que se tienen en la actualidad, como 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:



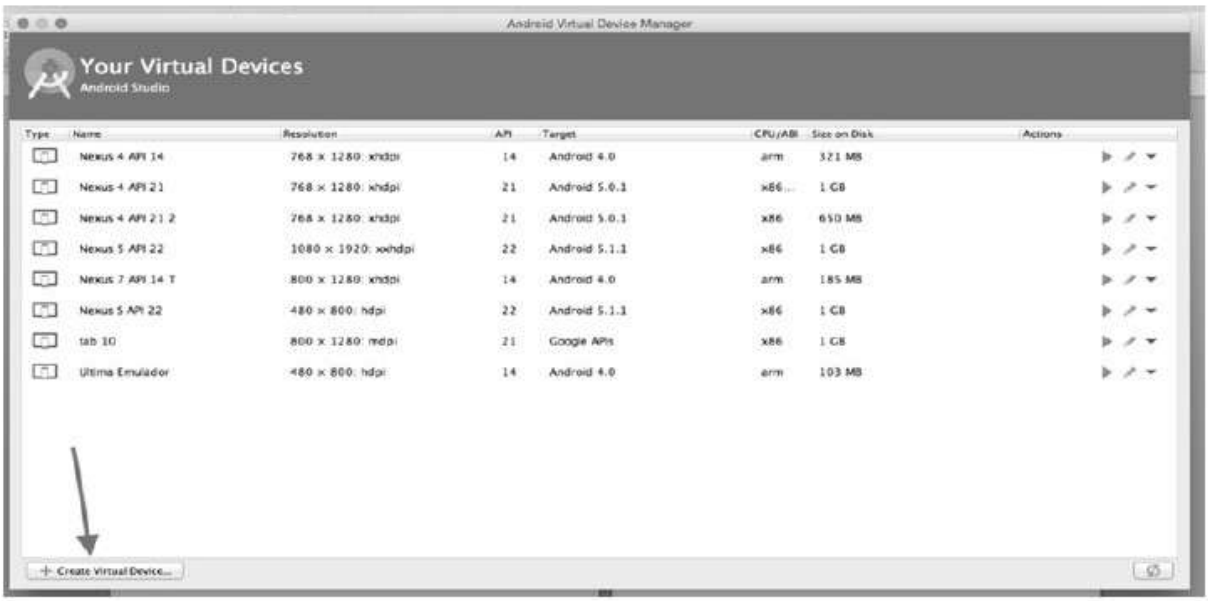
A la ventana se le conoce como AVD. El AVD significa “Android Virtual Device Manager” o en español, Administrador de dispositivos virtuales.

© Alfaomega - RC Libros

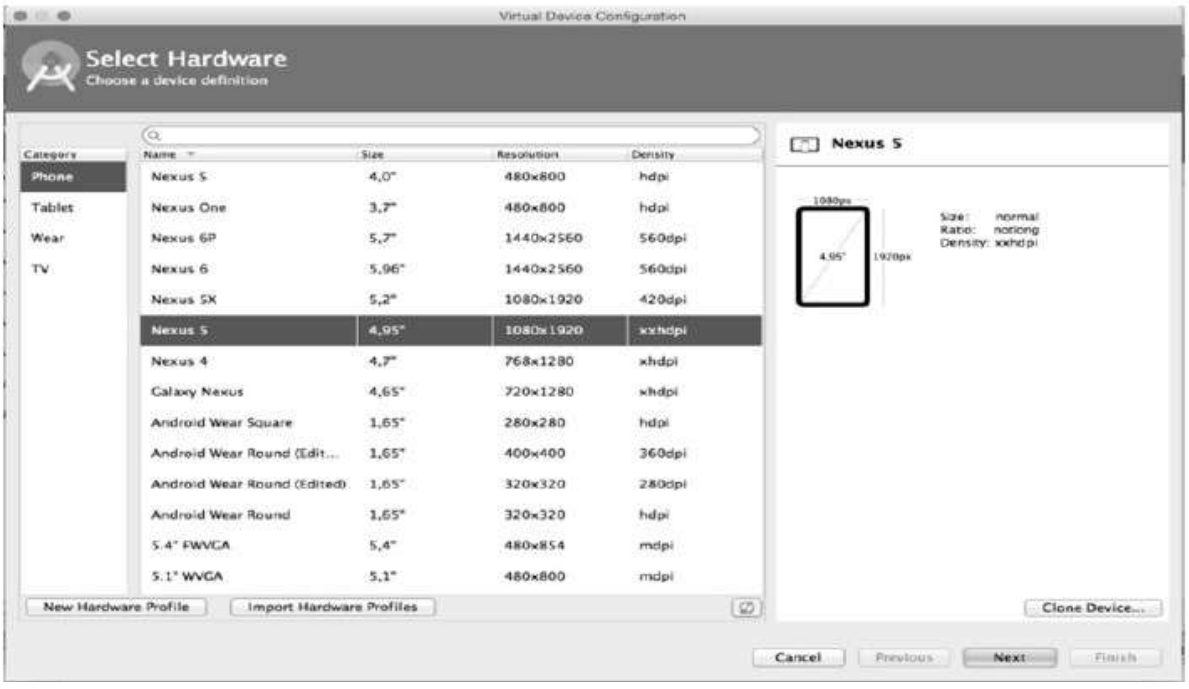
Esta ventana es la que nos permite administrar los dispositivos que ya tenemos preconfigurados, si es la primera vez que abres Android Studio no vas a encontrar dispositivos configurados y se va a ver totalmente blanco.

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:



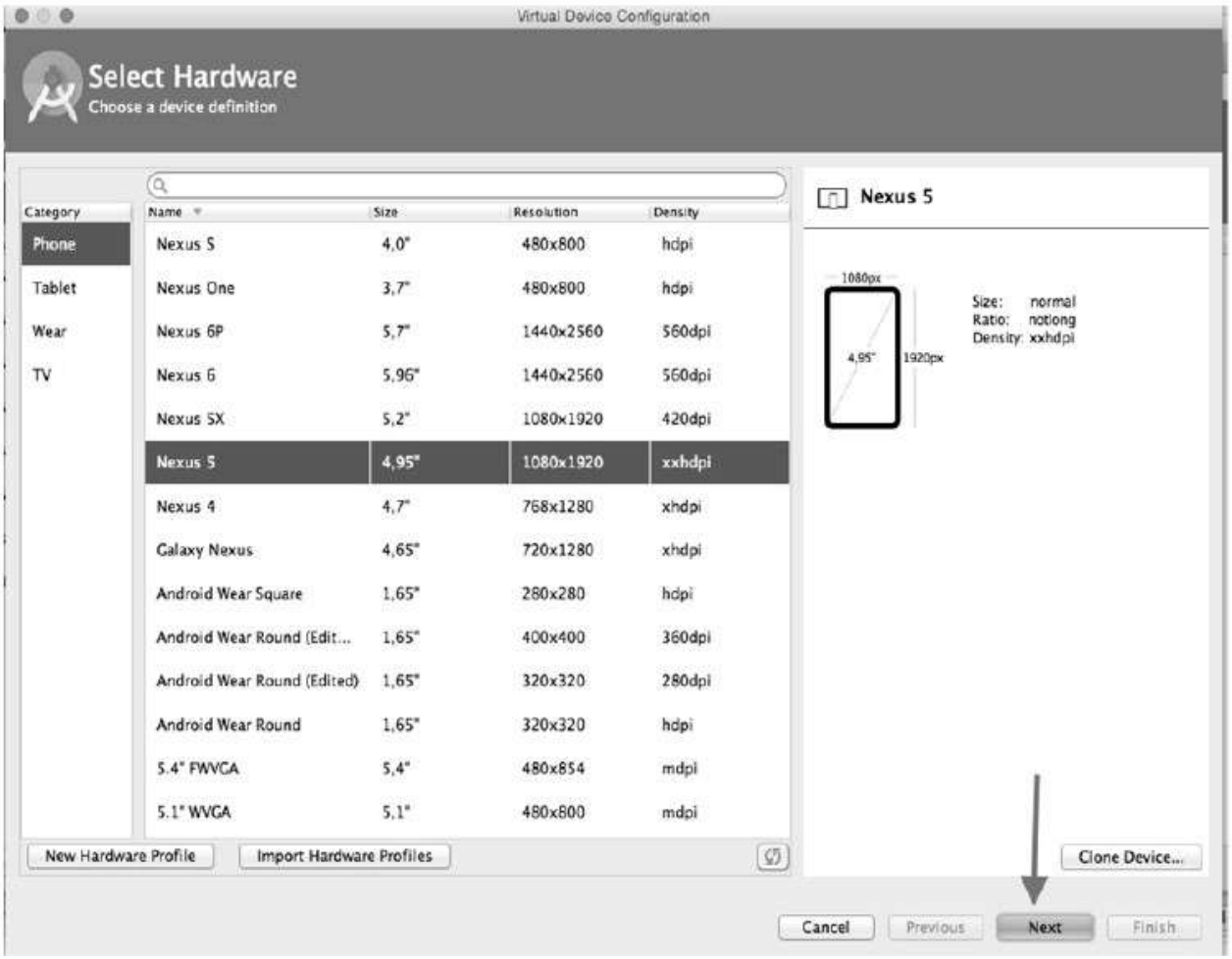
Podemos ver un menú principal que nos permite crear 4 tipos de dispositivos:

- 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.

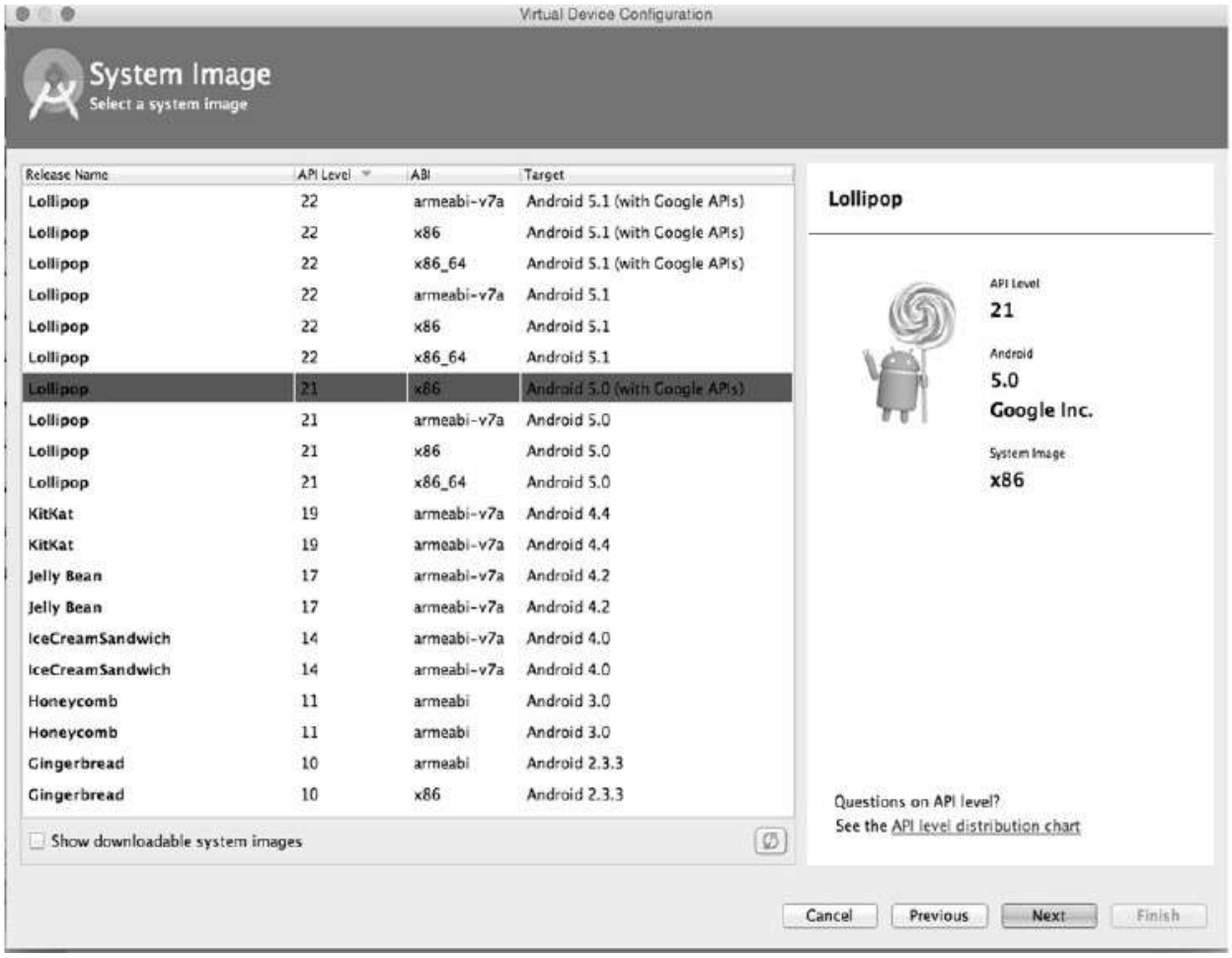
También te coloca algunas preconfiguraciones por tamaño de pantalla y/o por densidad, en nuestro caso vamos a crear de ejemplo un Nexus 5.



© Alfaomega - RC Libros

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.

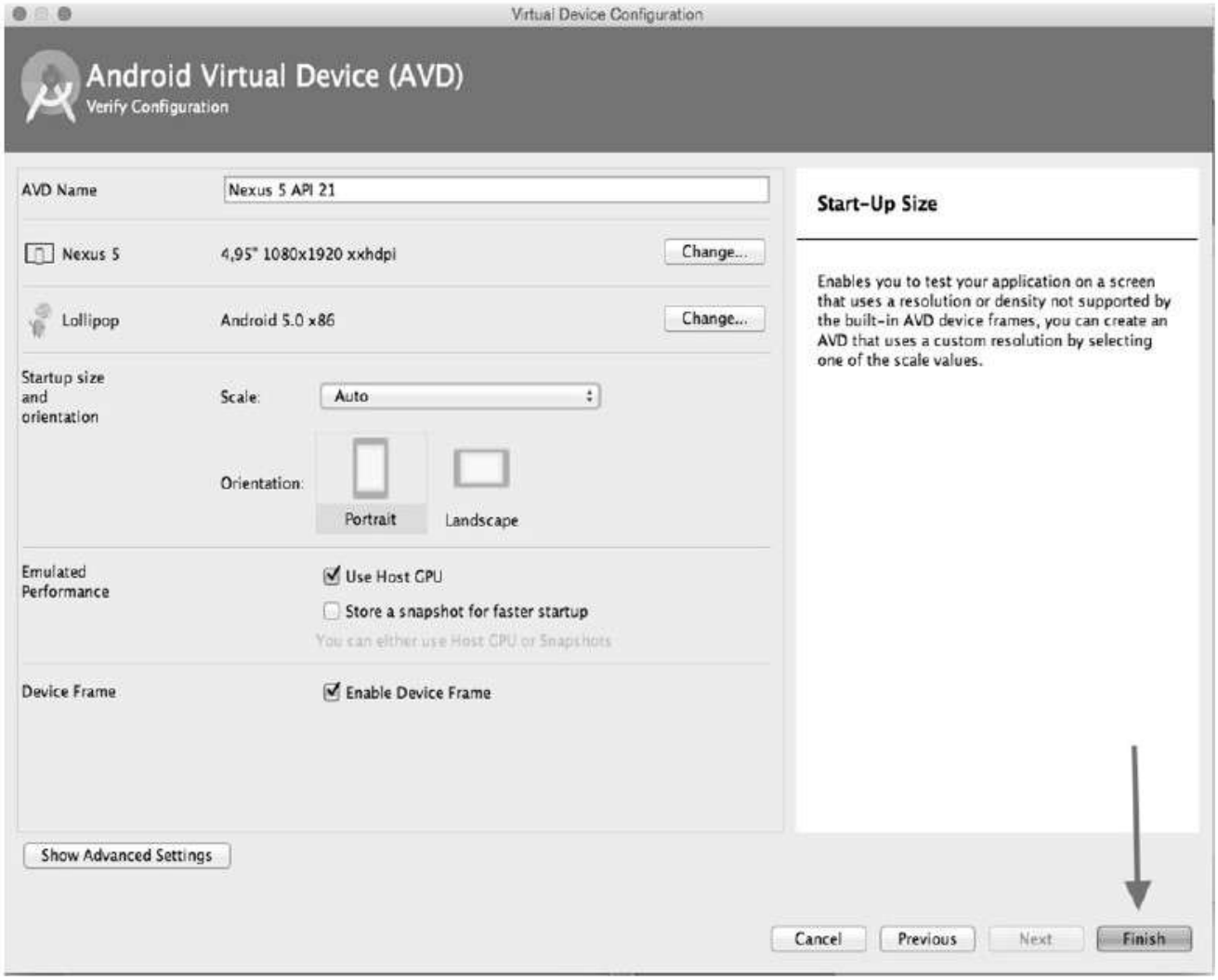
Scale: en esta opción podemos cambiar la escala para ver el dispositivo.

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.

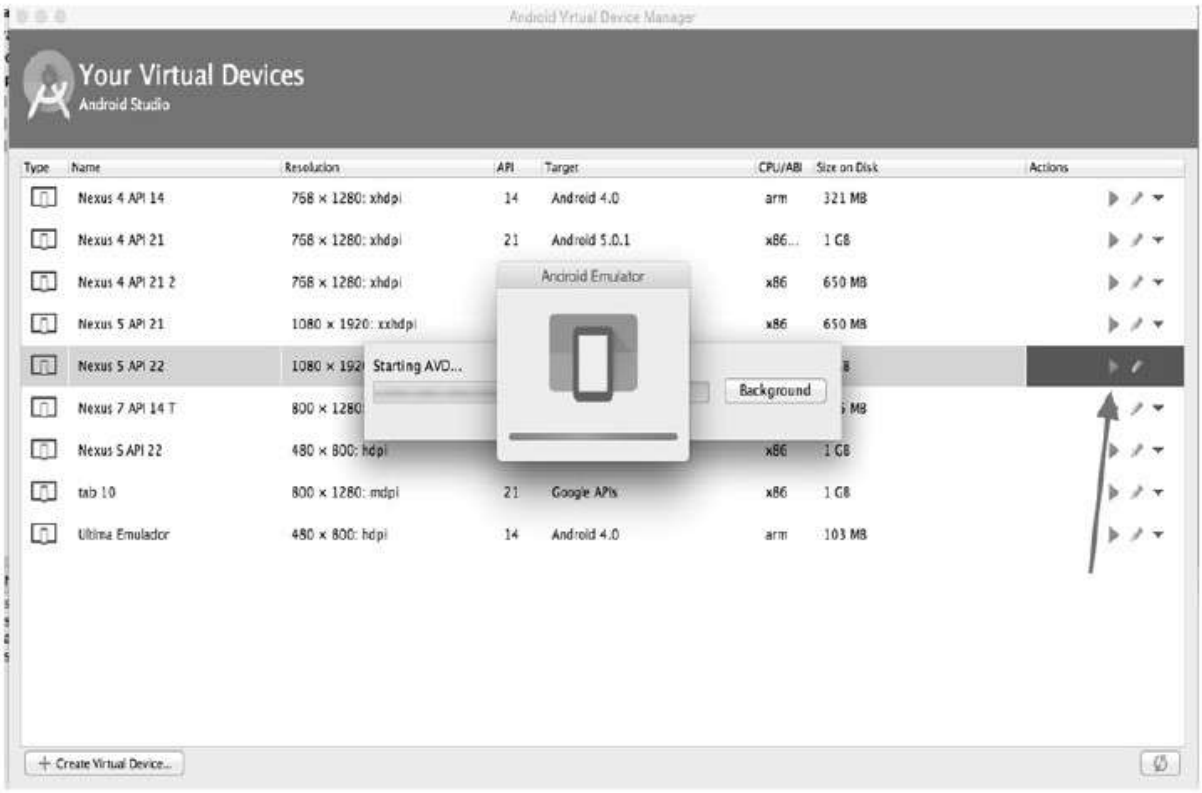
Ahora le damos clic en el botón Finalizar para acabar con la configuración.



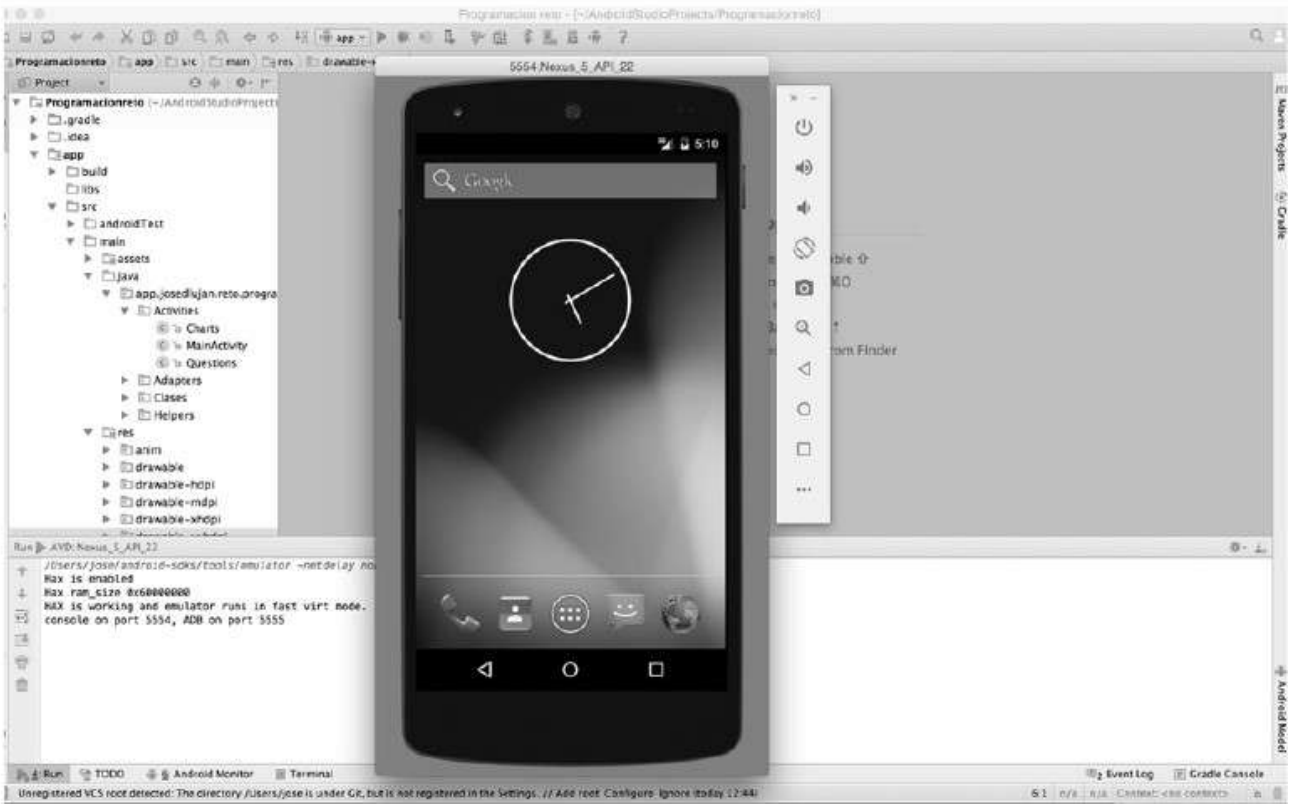
Con eso ya terminamos finalmente de configurar nuestro dispositivo.

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

© Alfaomega - RC Libros



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.

© Alfaomega - RC Libros

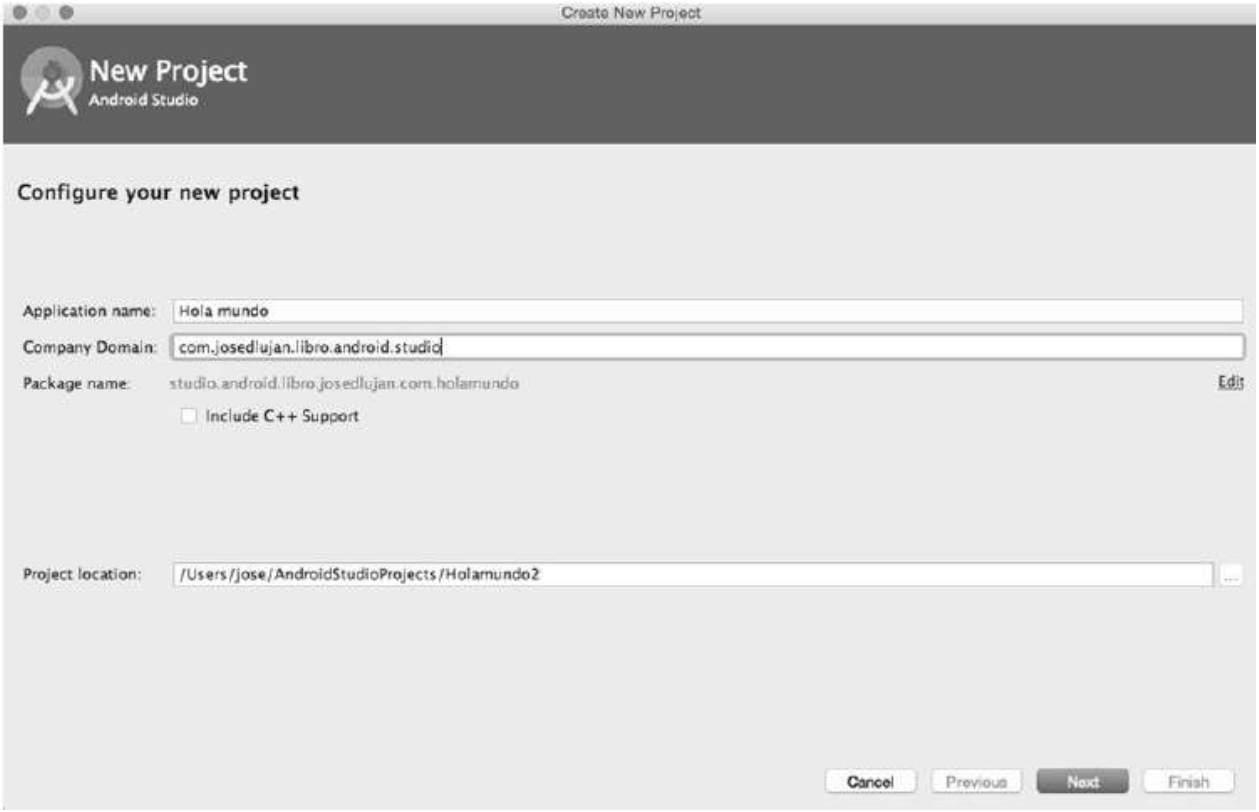
4 CREAR UNA APLICACIÓN

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.

© Alfaomega - RC Libros

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, Minimun 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:

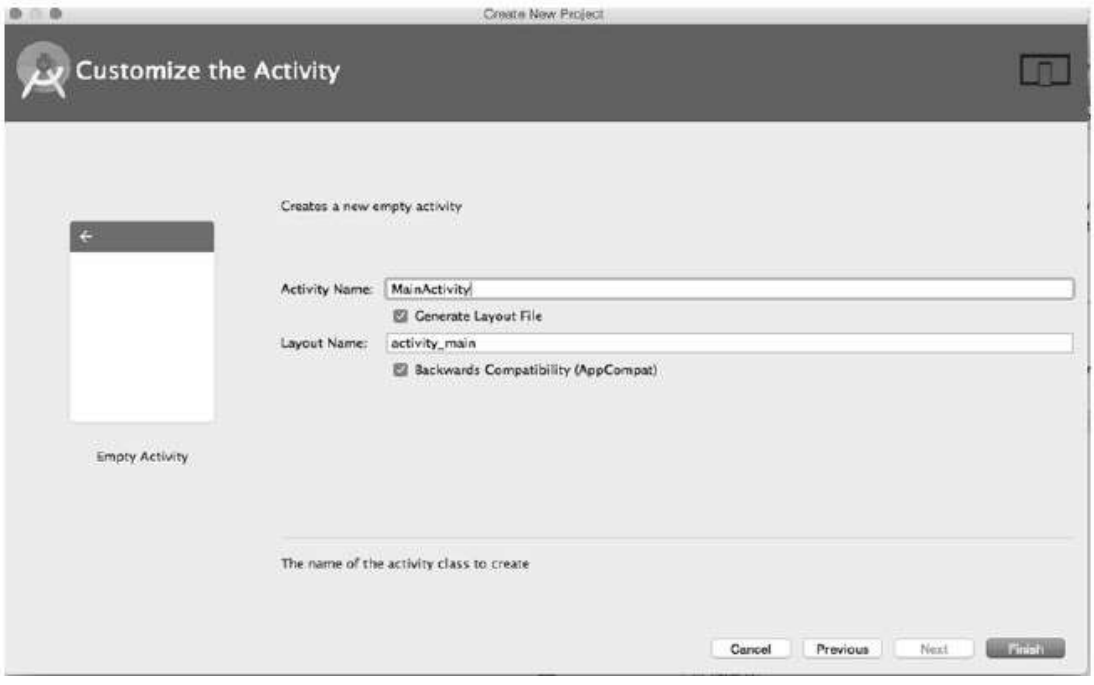
© Alfaomega - RC Libros



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:



© Alfaomega - RC Libros

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

Activity Name

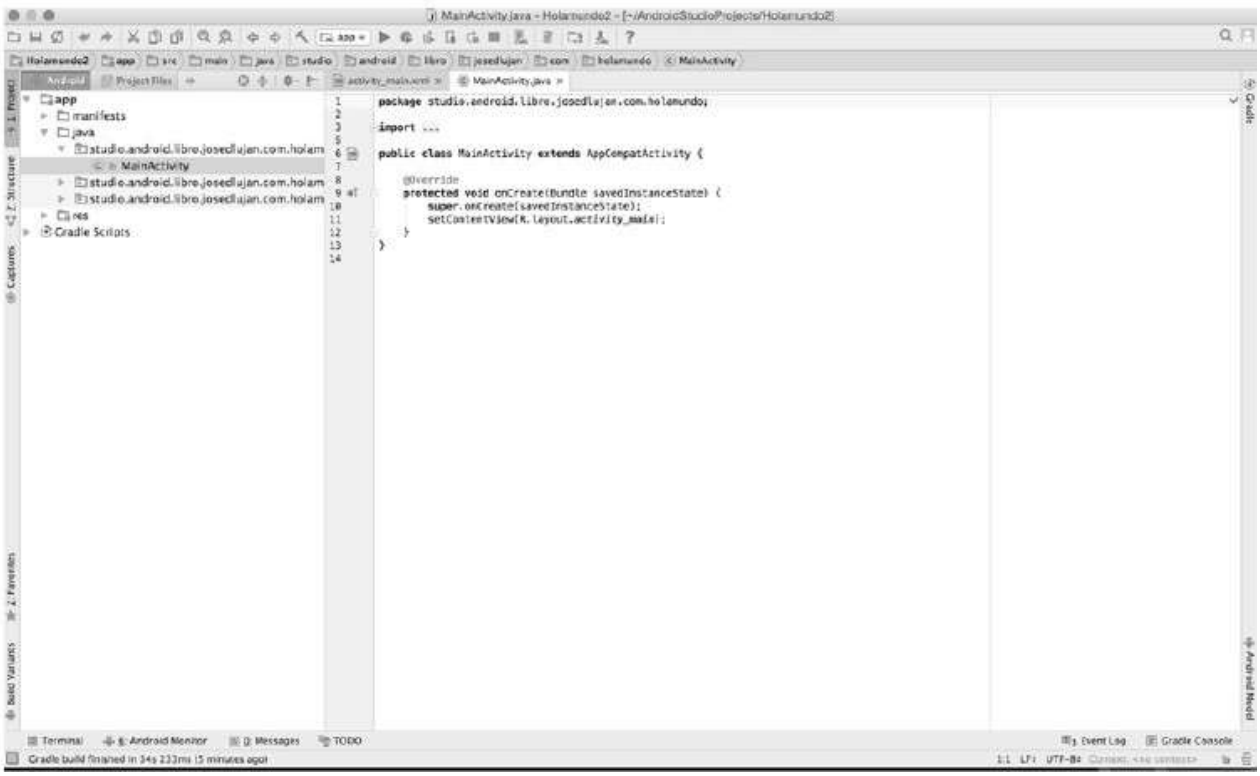
Es el nombre de la actividad y quiere decir que es el nombre de la clase que controla la primera pantalla, por convención la primera "activity" de una aplicación se llama MainActivity.

Layout Name

Este es el nombre de la maquetación 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:

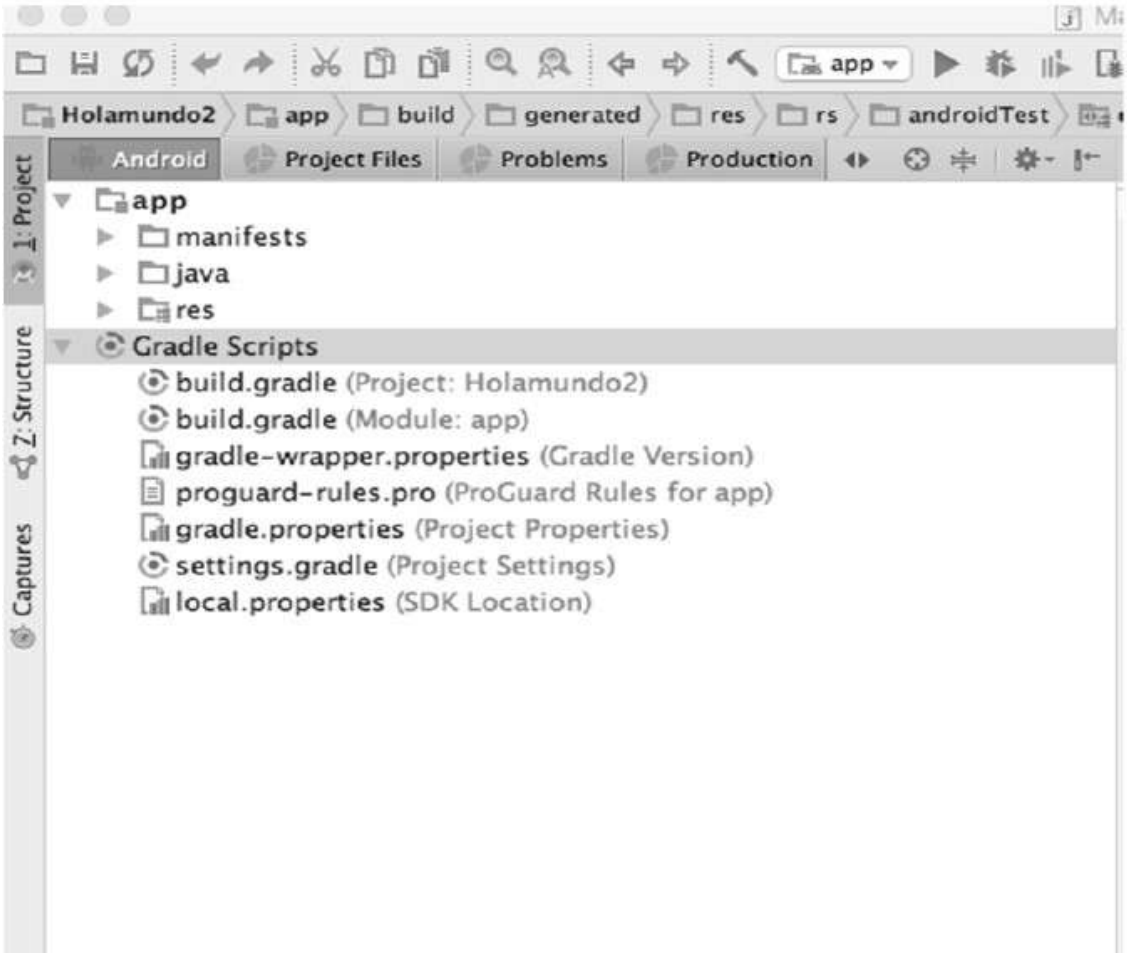


En realidad, para el "Hola Mundo" no tenemos que hacer nada en Android Studio, por defecto se crea esa instrucción y únicamente tuvimos que crear un proyecto vacío, con este terminamos el capítulo de la construcción de un "Hola Mundo".

© Alfaomega - RC Libros

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".*

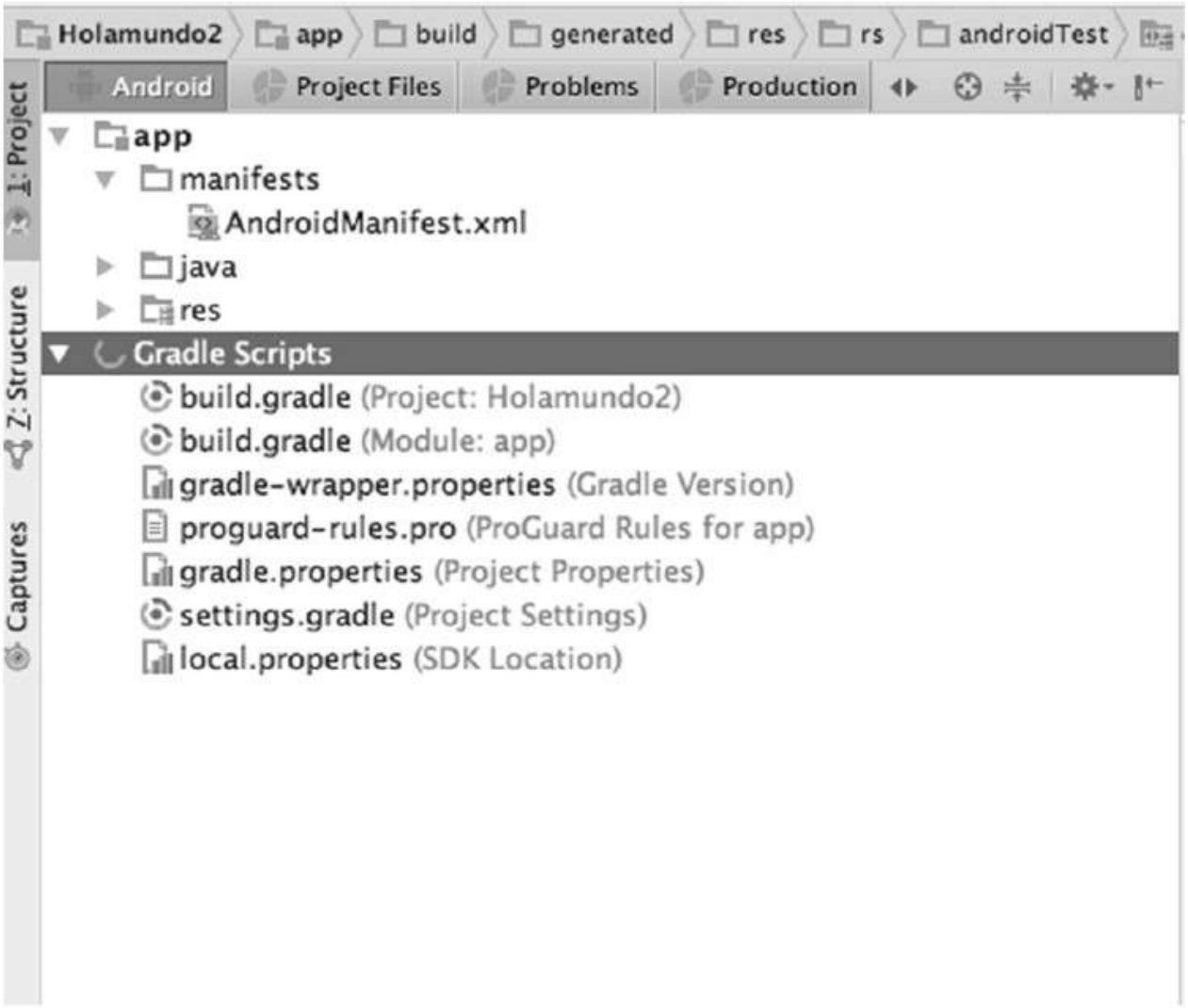
Dentro del directorio "app" tenemos una serie de directorios que podemos considerar como los principales dentro de una aplicación Android que son:

- Manifests
- Java
- Res
- Gradle Scripts

Directorio manifests

Dentro del directorio manifest podemos observar los siguientes archivos:

© Alfaomega - RC Libros



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"
```

© Alfaomega - RC Libros

```
      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"
```


© Alfaomega - RC Libros

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:

© Alfaomega - RC Libros

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

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

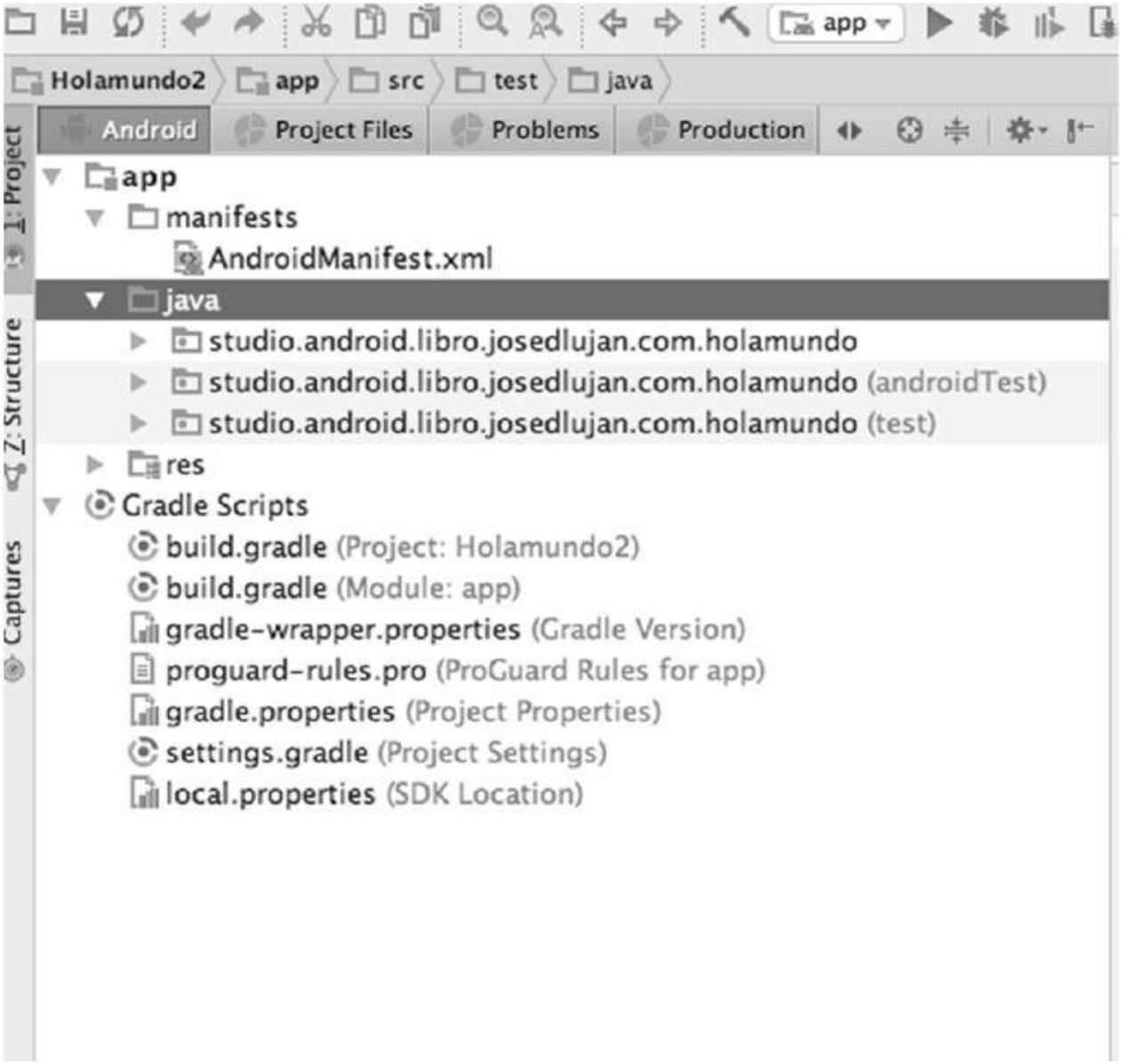
La primera línea indica el nombre del archivo Java que es la lógica de la aplicación, el nombre que tiene es el que le colocamos al inicio cuando creamos el proyecto que es “MainActivity”.

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

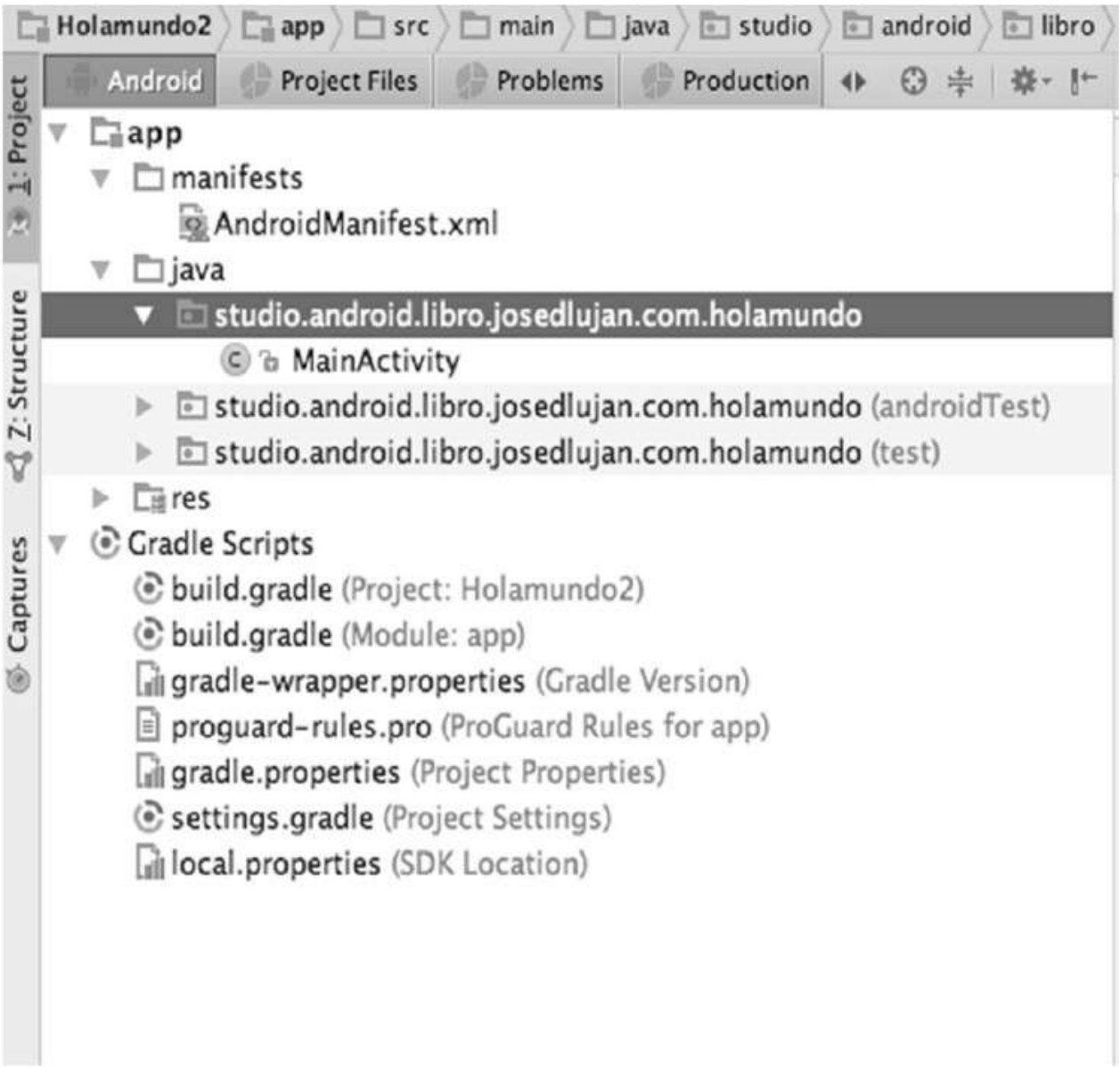
Si desplegamos el directorio Java veremos lo siguiente:

© Alfaomega - RC Libros



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:

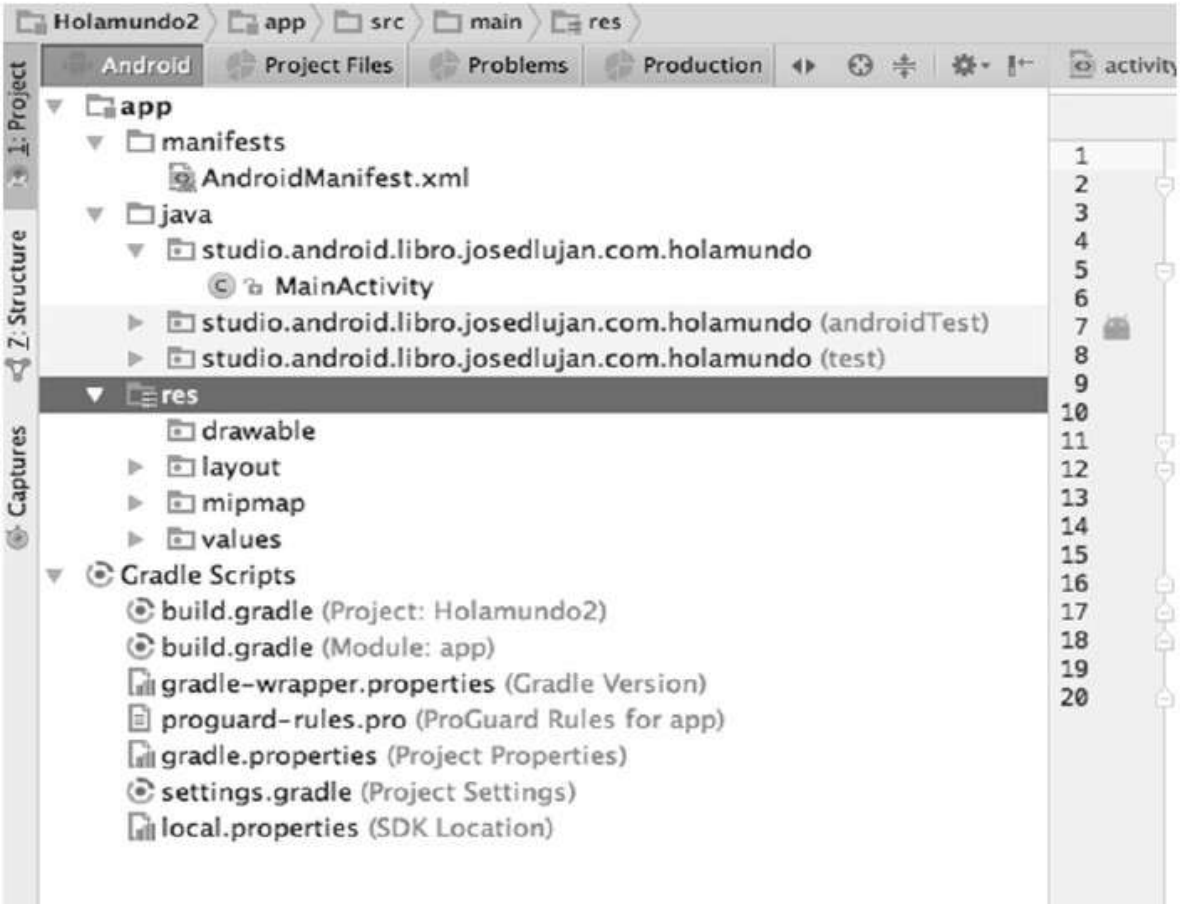


Directorio res

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:

© Alfaomega - RC Libros



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

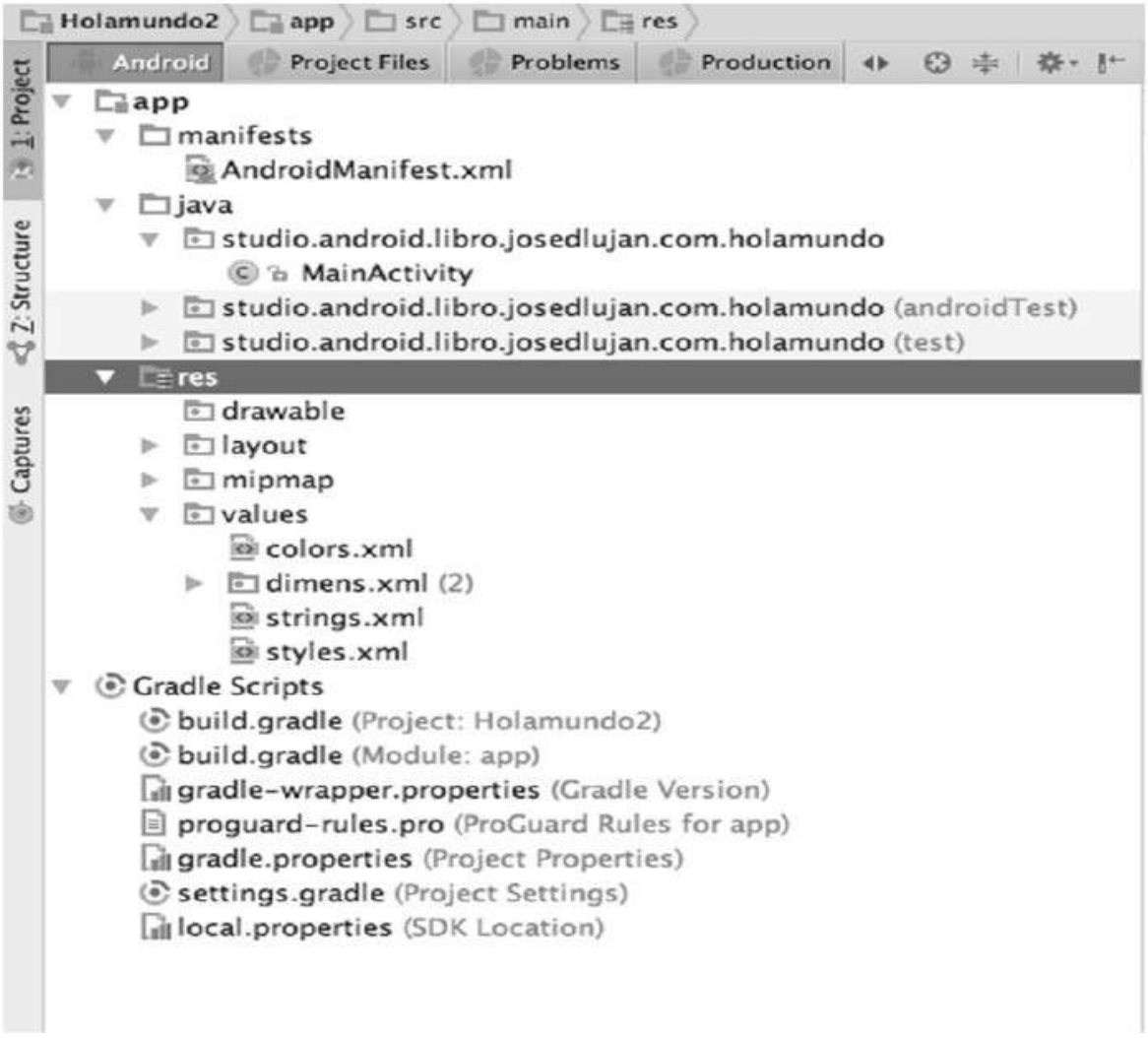
El directorio layout contiene todos los archivos que son las interfaces de las pantallas de la aplicación, por ejemplo: si tenemos actividades en nuestra aplicación, deberíamos de tener al menos 5 archivos xml que sean la interfaz de cada una de las actividades. Así que podemos decir que en este directorio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



© Alfaomega - RC Libros

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 "recomplicación", 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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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 ***tagetSdkVersion***. 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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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 actiivity_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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.MainActiv
ity">
</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

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.MainActiv
ity">


   <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

© Alfaomega - RC Libros

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>
```

© Alfaomega - RC Libros

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 strechColumns, 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 LinerLayout 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

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

8 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.

© Alfaomega - RC Libros

El código del ID sería el siguiente:

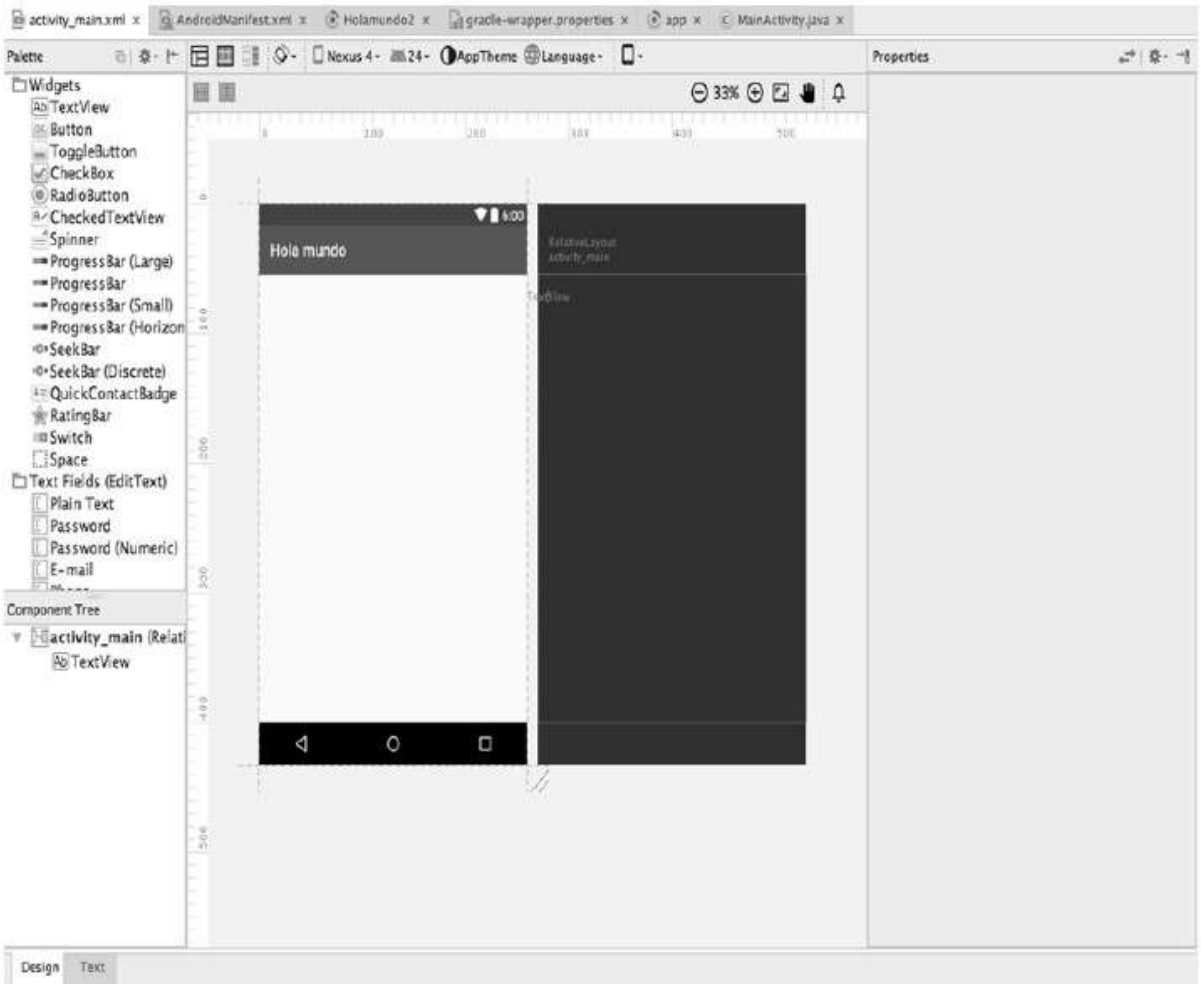
```
android:id="@+id/textView"
```

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"

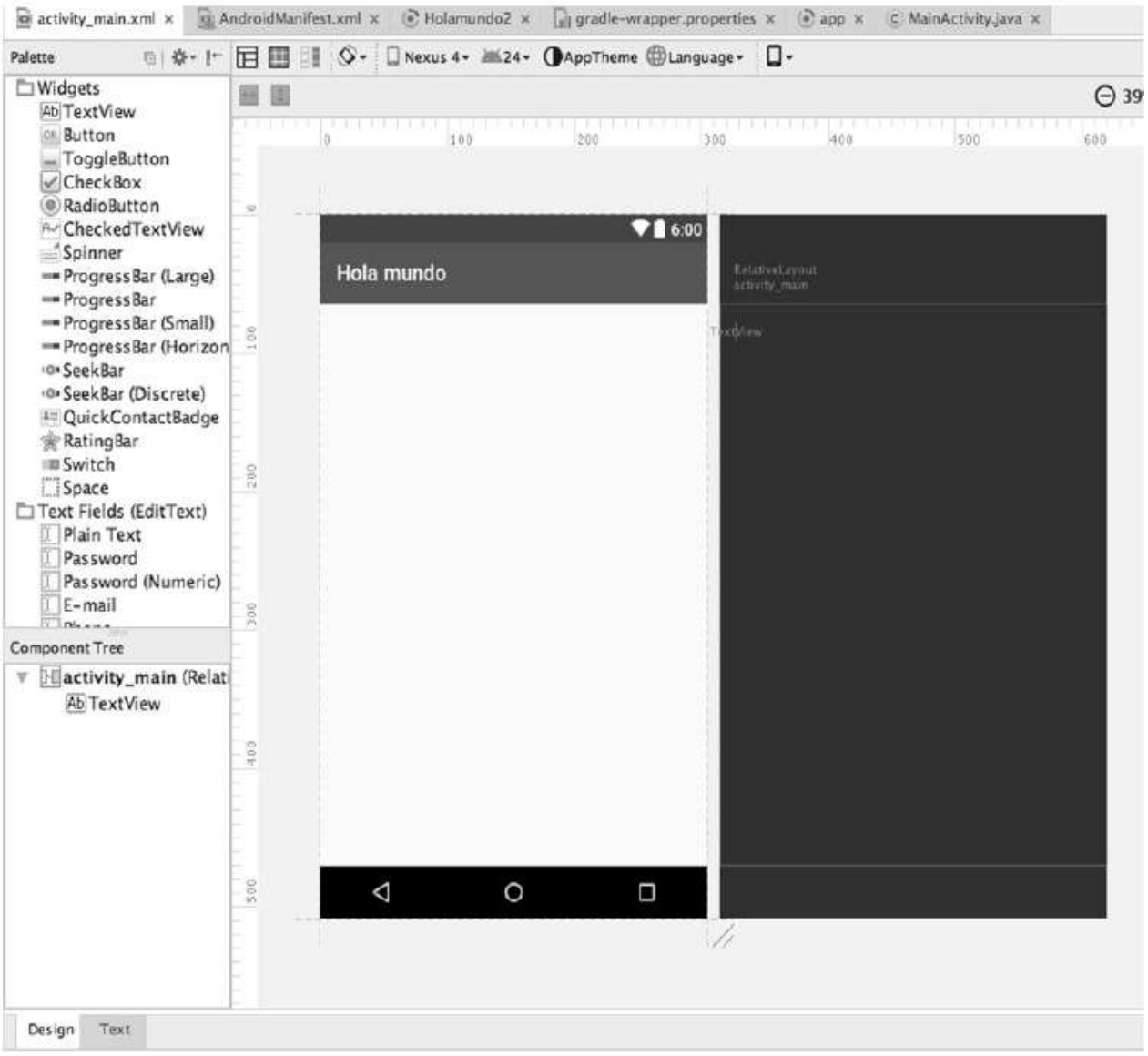
Esta pestaña por el contrario se basa en el código duro y puro, así que si se quiere colocar un elemento utilizando esta pestaña, lo que se va a observar es el código del archivo XML y se tendrían que agregar más líneas de código para el elemento. Este método es mucho más ligero de usar ya que no tiene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

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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"
```


© Alfaomega - RC Libros

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

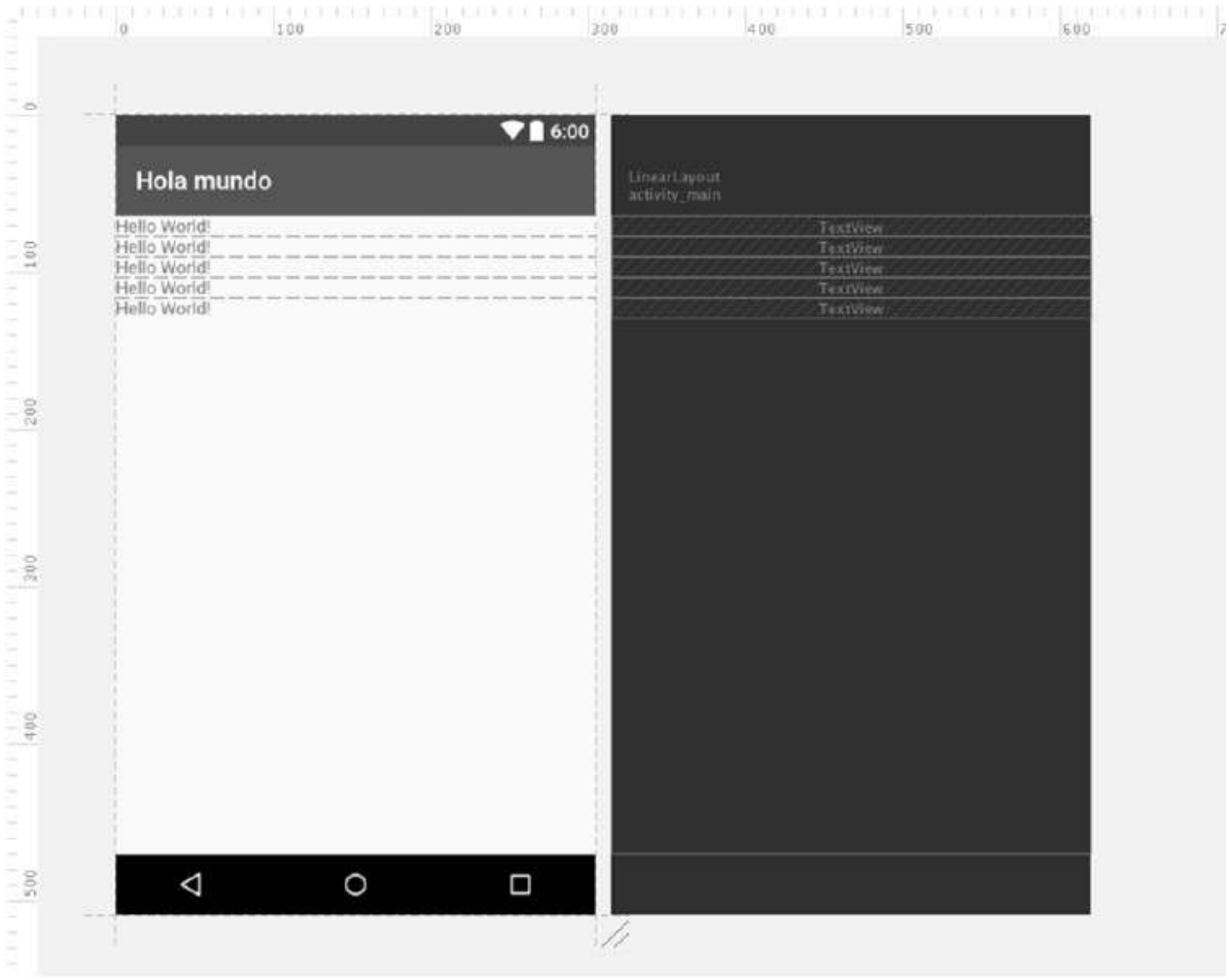
tools:context="studio.android.libro.josedlujan.com.holamundo.MainActiv
ity">


    <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:

© Alfaomega - RC Libros



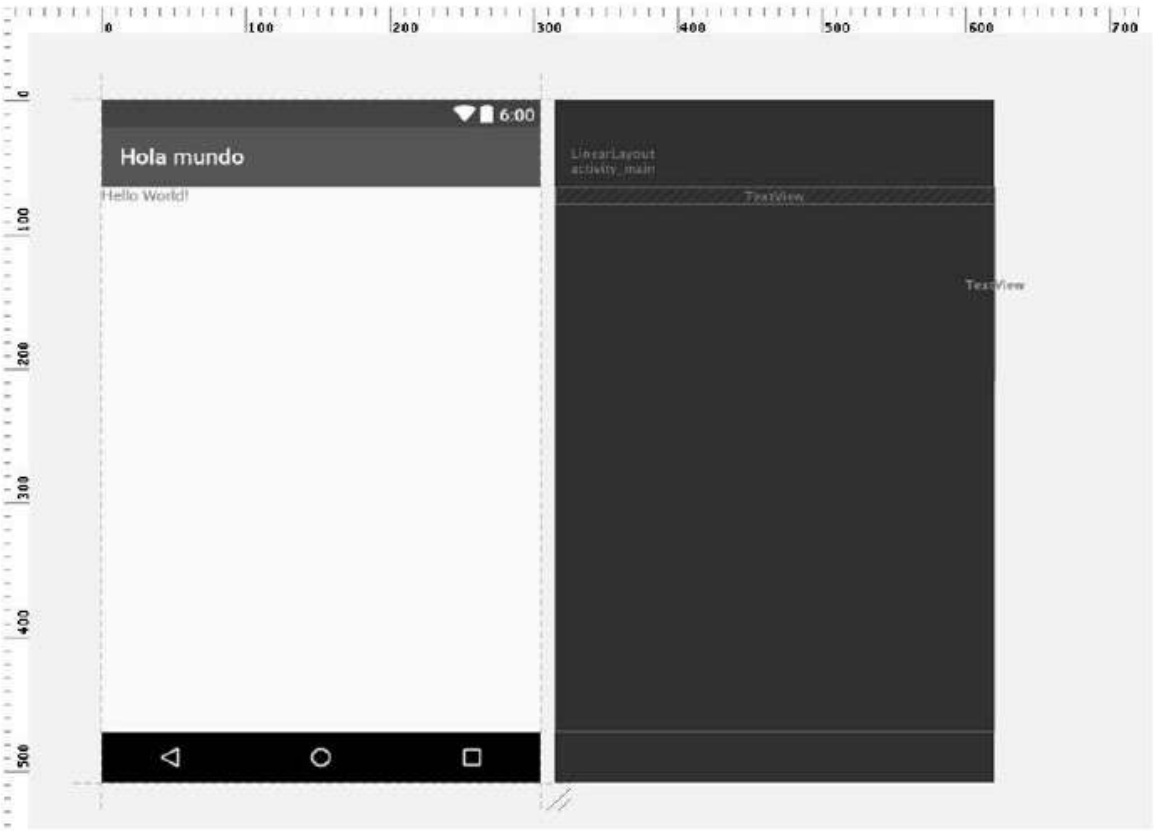
Los elementos textview están uno debajo del otro porque el contenedor padre, que es el linearlayout, colocó en su atributo orientation, que la orientación de los elementos que contendrá será de forma vertical, por lo tanto, no es importante el tipo de elemento, su tamaño o cualquier otro atributo, siempre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

© Alfaomega - RC Libros



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:

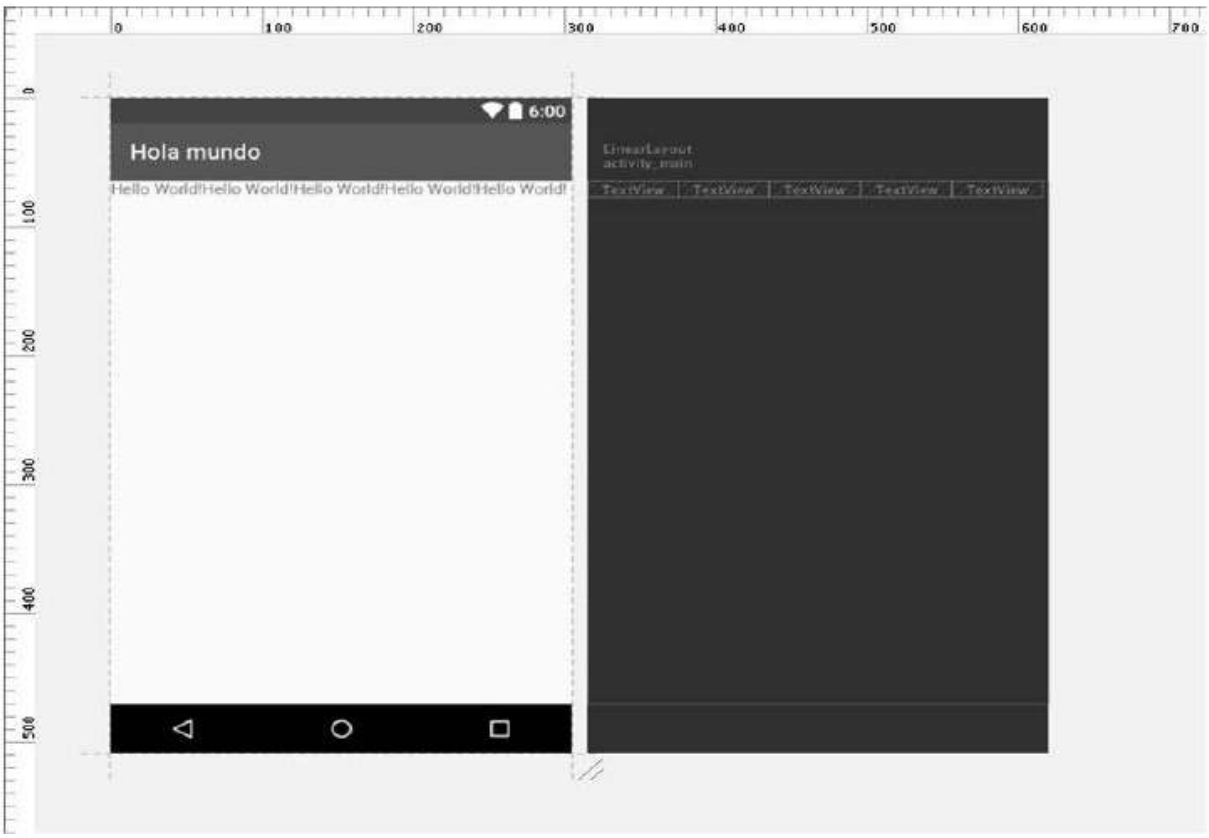
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"

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:



© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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>
```


© Alfaomega - RC Libros

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>
```


© Alfaomega - RC Libros

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"
    />
```

© Alfaomega - RC Libros

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"
```


© Alfaomega - RC Libros

```
    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:

© Alfaomega - RC Libros

```
    <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:

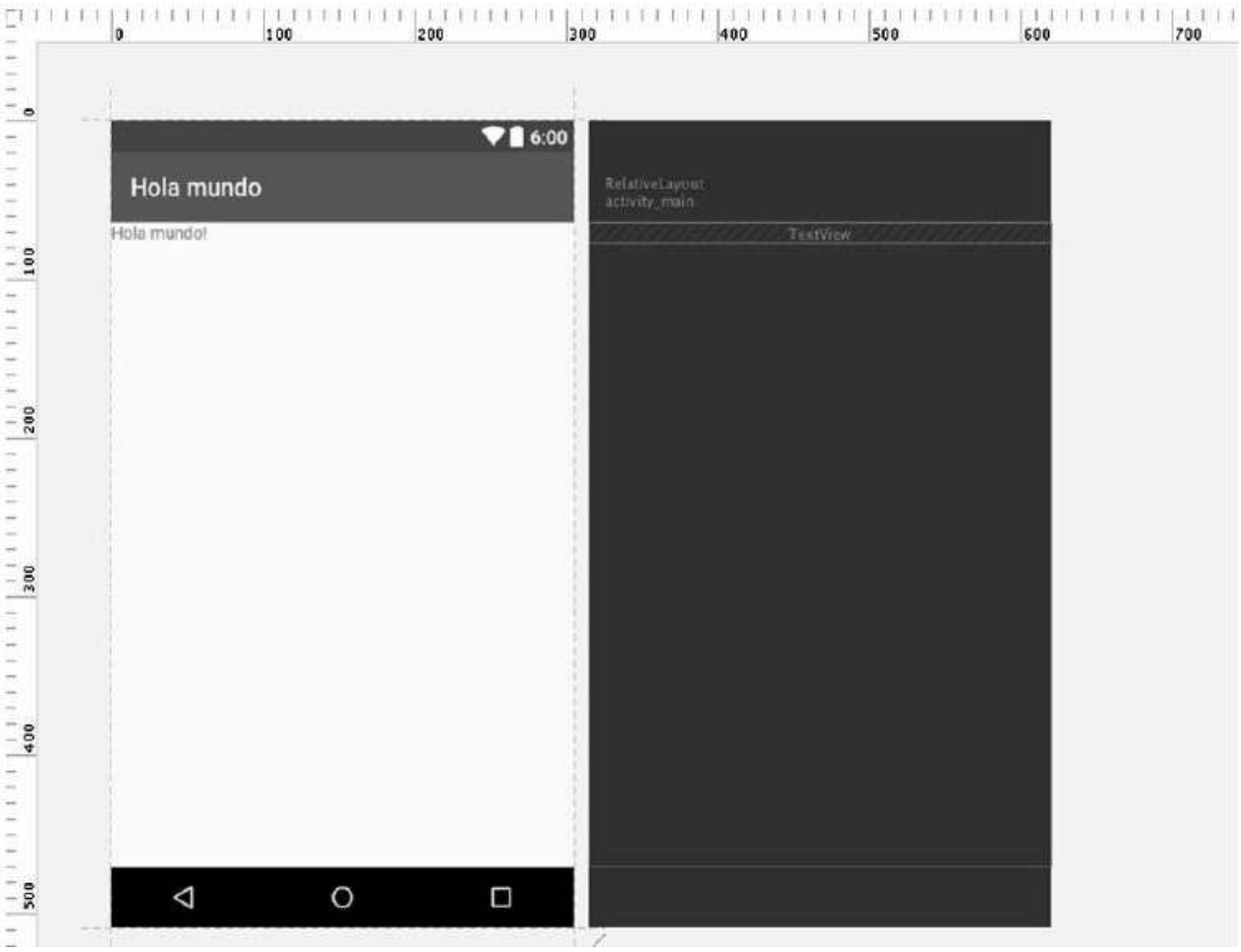
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">
```


© Alfaomega - RC Libros

```
<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"/>

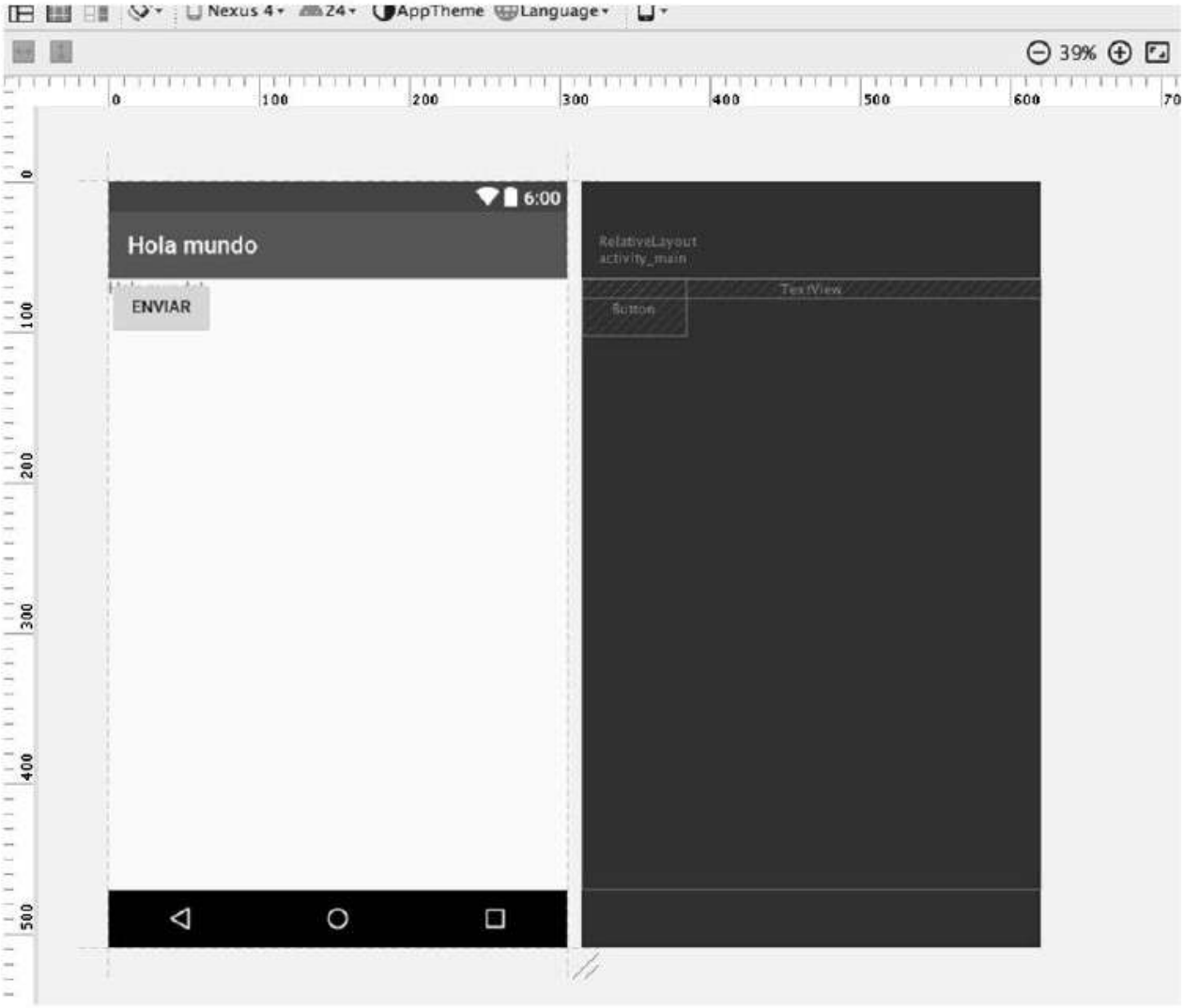

</RelativeLayout>
```

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:

© Alfaomega - RC Libros



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"
```

© Alfaomega - RC Libros

```
   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:

© Alfaomega - RC Libros



Podemos observar en la imagen que el textview se encuentra en la parte más arriba de la interfaz gráfica y que debajo del textview encontramos el botón que ya habíamos definido con el texto "Enviar". Esto lo tendríamos que aplicar para los distintos elementos que coloquemos.

Como parte de ejemplo vamos a colocar un botón más, 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:id="@+id/textview"
```

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

  <Button
    android:text="Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/enviar"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="148dp"

</RelativeLayout>
```

Las modificaciones en este código con respecto al anterior son sencillas, la primera es que el elemento botón ahora también cuenta con un identificador, el identificador tiene el nombre de enviar, es la primera línea del elemento:

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

El segundo elemento que agregamos tiene un id, aunque no nos sirve por el momento, cuenta con su alto y con su ancho como todo elemento y además con la propiedad below, que es para colocarlo después de otro elemento en este caso referencia el elemento enviar que es el primer botón que colocamos:

```
  <Button
android:text="Button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

Después tenemos la propiedad centerHorizontal, lo que hace esta propiedad es indicarle que vamos a colocar de manera centrada y horizontal el elemento y puede

© Alfaomega - RC Libros

tomar dos valores, falso o verdadero, en este caso lo colocamos como verdadero, en caso de no querer el acomodo simplemente no colocamos el atributo o podemos cambiar el valor a false, aunque por defecto si no le indicamos lo contrario, el valor es falso y por lo tanto no se acomoda centrado.

```
android:layout_centerHorizontal="true"
```

El siguiente atributo puede ser un viejo conocido para los desarrolladores que vienen del mundo web ya que el margin y el padding son elementos muy usados en la creación de una interfaz web, aquí el elemento puede de igual manera colocar de forma automática un margen a todo el elemento o a cada uno de sus lados, derecha, izquierda, arriba o abajo. En este caso el margen se coloca en la parte superior y la medida es de 148 dp, como hemos mencionado anteriormente las fuentes utilizan los SP, los espacios entre elementos o acomodo de elementos utilizan los DP. Así que siempre que colocamos un margin o padding lo ideal es utilizar la medida de dp.

```
android:layout_marginTop="148dp"
```

La representación visual de lo que hicimos se debe de ver de la siguiente forma:



Las propiedades *padding* sirven para rellenar, funcionan de la misma manera que se usa el *padding* en *html*, reservan un espacio en el contenedor. El *padding* no es exclusivo del *RelativeLayout*, también funciona para el *LinearLayout* y muchos otros elementos gráficos de Android, pero el "holamundo" en Android tiene esta estructura para que la cadena "holamundo" no esté pegada a la parte superior e izquierda del layout.

Margin y Padding

Estas propiedades son viejas conocidas del desarrollo web, así que si ya tienes experiencia en el mundo de los sitios web podrás asimilar toda esta información de manera rápida, en caso contrario solamente es cuestión de practicar un poco más y listo.

Comenzaremos realizando un ejemplo con Margin ya que es mucho más fácil de comprender, para eso partiremos de tener un linearlayout con dos elementos del tipo botón. 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">

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

   <Button
      android:text="Button"
      android:layout_width="wrap_content"
```


© Alfaomega - RC Libros

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

</LinearLayout>
```

El primer bloque es un simple linearlayout:

```
    <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 es un botón que solo contiene texto y su alto y ancho.

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

El tercer bloque es el segundo botón con los mismos elementos que el primero, pero la diferencia es el texto, en este caso dice "Button".

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

La representación visual es la siguiente:



Podemos notar que los 2 botones se encuentran en la misma línea y con muy poca distancia entre ellos, si quisiéramos colocar distancia de por medio podríamos colocarle la propiedad margin al primero o al segundo, dependiendo de los otros elementos que podemos tener alrededor, pero en este caso como solamente tenemos estos dos elementos nos da lo mismo a quién se la coloquemos.

Así que se la vamos a colocar al primer botón, al que tiene el texto enviar y el código quedaría de la siguiente forma:

```
<Button
   android:text="Enviar"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginRight="100dp"
   />
```


© Alfaomega - RC Libros

Solamente modificamos el elemento botón que tiene el texto "Enviar", por lo tanto solo tenemos que poner atención en ese código, vemos que la última línea es la que tiene la instrucción marginRight, en donde le decimos que el espacio que va a colocar es el de 100 dp. Por lo tanto, ese espacio se va a colocar de por medio con cualquier elemento a la derecha.

Visualmente veríamos este cambio:



Podemos observar que la distancia entre los botones aumentó considerablemente, así que un medio para colocar espacios o agregar distancia entre

los elementos es el uso del margin, si utilizamos la medida dp para asignar los espacios no deberíamos de tener ningún problema con mantener la misma proporción de espacio sin importar los tamaños de pantalla o las densidades.

Ahora vamos con el elemento padding.

Esta propiedad también nos sirve para mantener un espacio, pero podemos decir que padding es internamente y margin es externamente, lo que vamos a hacer en este caso es aplicar la propiedad padding pero al elemento linearlayout, el elemento que es el contenedor padre de esta interfaz.

Partimos del último código que hemos realizado que 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">

    <Button
        android:text="Enviar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="100dp"
        />

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

</LinearLayout>
```


© Alfaomega - RC Libros

La representación visual es esta:



Ahora vamos a modificar el linearlayout agregando la propiedad padding y asignando una medida, el código queda de la siguiente forma:

```
<?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"
    android:padding="50dp"

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


    <Button
        android:text="Enviar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="100dp"
        />


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

</LinearLayout>
```

La única línea que nos interesa para este caso es la línea del LinearLayout que tiene la instrucción padding:

```
android:padding="50dp"
```

Esta línea indica que necesitamos un espacio de 50 dp hacia todos los lados, lado izquierdo, derecha, arriba y abajo, recuerda que le podemos colocar de forma especifica solamente un lado, pero en este caso lo hicimos para todos lados, así que es como si fuera el marco de una puerta, todos los datos deben estar libres 50 dp, así que vamos a ver el cambio en la posición de los botones que ya tenemos. El cambio se verá así:

© Alfaomega - RC Libros



Como vemos el movimiento es bastante notorio ya que del lado superior del layout tenemos 50 dp libres, pero también del resto de los lados, así que el padding es para despejar una posición que necesitemos para lograr un mejor acomodo de los elementos, si usamos los dp como medidas no tendríamos que tener problemas en que nuestra aplicación se vea siempre con las mismas proporciones si respetamos esto.

Pesos y distribución

Los pesos y la distribución de los elementos en Android son algo muy sencillo de entender pero que a la mayoría de los que se inician en el mundo del desarrollo Android les cuesta un poco por la parte del mundo web y las limitaciones que cuenta XML. Los pesos son la forma de distribución más sencilla para poder trabajar una interfaz en Android, supongamos que tenemos 2 botones y queremos que esos botones ocupen la mitad de la pantalla cada uno. Este sería nuestro código:

```
<?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"
   android:weightSum="2"

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

   <Button
      android:text="Boton 1"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      />
   <Button
      android:text="Boton 2"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      />

</LinearLayout>
```

Podemos ver que en el elemento padre tenemos una línea nueva de código que no hemos mencionado hasta este apartado en el libro que es weightSum, esta instrucción nos sirve para indicar que en el elemento padre tenemos una proporción

© Alfaomega - RC Libros

de 2. Este tema de los pesos es matemática pura y sencilla. Si el elemento principal contiene 2 como unidad y un elemento hijo tiene el peso de 1 y el otro elemento hijo tiene el peso de 1, es decir, que estamos distribuyendo en total del ancho en 2 partes iguales, en otras palabras, sería decir que damos el 50% a uno y el 50% a otro. La línea de código que marca el total que tenemos para distribuir es esta:

```
android:weightSum="2"
```

Ahora tenemos el primer botón que tiene dos líneas que nos deben llamar la atención, la línea del ancho que tiene como ancho 0 dp, al decir que el ancho es 0 dp estamos renunciando a que el ancho de este elemento se pueda obtener de esta línea, el ancho se va a obtener de otra parte:

```
android:layout_width="0dp"
```

La línea que nos va a servir como referencia ahora para el ancho es la línea del peso que es:

```
android:layout_weight="1"
```

Como vemos le estamos asignando el peso de 1 y como nuestro contenedor padre tiene el peso de 2, en otras palabras, le damos el 50%.

Podemos ver que nuestro otro elemento botón también tiene las mismas características:

```
  <Button
android:text="Boton 1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
/>
```

Este será el resultado de aplicar los pesos iguales:



Ahora vamos a hacer un ejercicio con diferentes proporciones para dejar claro el tema de los pesos ya que es importante tener muy aprendido este concepto.

Vamos a crear 3 botones, uno que use el 25% de la pantalla, otro que use el 25% también, y finalmente uno que utilice el 50% de la pantalla, sumados juntos dan el 100% de ancho de la pantalla. Este sería el código:

```
<?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"
```


© Alfaomega - RC Libros

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:weightSum="4"

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

    <Button
        android:text="Boton 1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        />
    <Button
        android:text="Boton 2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        />
    <Button
        android:text="Boton 3"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        />

</LinearLayout>
```

Podemos ver que en nuestro linearlayout tenemos un nuevo valor para repartir que es el valor de 4:

```
android:weightSum="4"
```

El botón 1 tiene la misma indicación en el peso de 0 dp para que ignoremos el ancho de ese atributo y tomemos en cuenta como el ancho el del atributo layout_weight, el ancho que le asignamos es el de 1 y si hacemos el cálculo matemático podemos ver que 1 sería el 25% de 4:

© Alfaomega - RC Libros

```
    <Button
android:text="Boton 1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
/>
```

Ahora tenemos el botón 2, este botón tiene las mismas propiedades que el botón 1 ya que tenemos la intención de que también tenga un área del 25%, la misma que el primer botón:

```
    <Button
android:text="Boton 2"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="1"
/>
```

Llegamos al último elemento en el que este por sí solo ocupará o debe de ocupar el 50% de la pantalla, así que si tenemos el valor de 4 como el 100% deberíamos de asignarle el valor de 2 para que este sea el 50%.

```
    <Button
android:text="Boton 3"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_weight="2"
/>
```

Ahora veremos el resultado en nuestra pestaña de diseño en Android Studio:

© Alfaomega - RC Libros



Hemos logrado lo que nos propusimos al principio de este ejercicio que era lograr 3 botones, uno con el 25% de la pantalla, otro con el 25% también y finalmente un tercero con el 50% por sí solo. El peso es una ayuda muy eficaz para lograr que nuestras aplicaciones se vean de igual forma sin importar el tamaño del dispositivo, la marca, etc. Aplicar los porcentajes es la mejor forma que tenemos para desarrollar una aplicación funcional y bien hecha para Android.

9 DENSIDADES Y TAMAÑOS DE PANTALLA

En los inicios de Android no era un tema muy llamativo el de las densidades de pantallas, ya que existían muy pocos modelos, fabricantes y características, pero hoy en día el tema de las densidades es bastante complejo para el usuario común ya que muchos de los conceptos que se utilizan tienen relación con elementos que no se pueden ver a primera vista o que no van del todo bien con alguien que no está familiarizado con la tecnología.

Densidades

En Android contamos con 6 densidades de pantallas.

- ldpi – densidad baja
- mdpi – densidad media
- hdpi – densidad alta
- xhdpi – extra alta densidad
- xxhdpi – extra extra alta densidad
- xxxhdpi – extra extra extra alta densidad

En valores numéricos cada densidad representaría la siguiente equivalencia:

- ldpi 120 dpi
- mdpi 160 dpi
- hdpi 240 dpi
- xhdpi 320 dpi
- xxhdpi 480 dpi
- xxxhdpi 640 dpi

Esto es muy probable que el siguiente año cambie y se agregue una nueva densidad, las densidades no tienen una relación con el tamaño de pantalla a diferencia de lo que se cree.

Tamaños de pantalla

Podemos tener 4 tamaños de pantalla hoy en dia en el mundo Android.

- Pequeño
- Normal
- Largo
- Extralargo

Un tamaño pequeño de pantalla va de las 2 pulgadas a las casi 3,7 pulgadas, el tamaño normal va de las 3,5 pulgadas a las 4,7 pulgadas, un tamaño largo de pantalla va de las 4,2 pulgadas a las 7 pulgadas y un tamaño extra largo va de las 7 pulgadas en adelante.

Hay dispositivos que quedan en rangos intermedios; por ejemplo, un dispositivo con 3,6 pulgadas entra en 2 rangos, en el pequeño y en el normal. Como desarrolladores no tenemos que preocuparnos tanto, ya que si creamos bien nuestra aplicación y todos los recursos establecidos están bien realizados para el tamaño pequeño y normal, va a tener un funcionamiento correcto. El fabricante y Android se encargan de colocarlo en una categoría en relación con el hardware y otras características.

Como puedes ver el tamaño de pantalla es mucho más fácil de definir ya que está en pulgadas, algo mucho más sencillo de analizar y comprender.

Relación densidad y tamaño de pantalla

En realidad, no es una norma que cuanto mayor tamaño tenemos también mayor densidad, el usuario promedio puede creer que tener una tablet de 10 pulgadas tiene una mejor calidad que un teléfono de 7 pulgadas, pero depende de muchas cosas, por ejemplo en el caso de una imagen, si se tomó una imagen de mala calidad se va a ver igual de mal en un teléfono o una tablet sin importar el tamaño de pantalla o la densidad, ya que la baja calidad viene en el archivo y no en el dispositivo.

© Alfaomega - RC Libros

En el caso de que necesitemos comparar dos pantallas y ver cuál es mejor, el tamaño de pantalla pasa a segundo término, claro que si el usuario es una persona débil visualmente o con un problema de la vista es de gran ayuda el tamaño de la pantalla, pero a nivel de calidad la pantalla de menor tamaño puede tener una mejor calidad de imagen, esto es dependiendo de su densidad.

Cuanta mayor densidad tenga un dispositivo mayor calidad tiene su pantalla, pero es importante recordar que si la imagen es de baja calidad no importa que tengas el dispositivo con la más alta densidad, se verá igual de mal sin importar el dispositivo.

Regresando al tema de la densidad, hoy en dia las 2 densidades con mayor calidad son las xxhdpi y xxxdhpi, que son las densidades que hoy cualquier teléfono de gama alta o algún superdotado de gama media puede llegar a tener.

La densidad se obtiene dependiendo de los píxeles que se tienen por pulgada, veamos un ejemplo sencillo.

La densidad de la población se calcula haciendo el cálculo de cuánta gente tenemos en una ciudad por metro cuadrado, podemos decir que si en una ciudad tenemos 2 habitantes por metro cuadrado y en otra tenemos 6 habitantes por metro cuadrado la que tiene 6 es la ciudad más densamente poblada, ya que tiene más habitantes por metro cuadrado. Sería lo mismo pero aplicado a pantallas. En lugar de personas hablemos de píxeles y en lugar de metros cuadrados hablemos de pulgadas, con eso creo que queda explicado lo de la densidad de pantalla, a mayor cantidad de píxeles en una pulgada mayor es la densidad y mayor es la calidad ya que podremos dibujar más colores.

Aquí viene una de las relaciones raras entre densidad y tamaño, cuanto más grande es la pantalla de un teléfono también podemos decir que se pierde calidad, ya que se distribuyen en una manera más amplia los puntos y se va perdiendo densidad, tendríamos entonces que colocar más puntos dentro de una pantalla más grande para mantener la densidad, obviamente esto lo tendría que hacer el fabricante ya que este cambio es físico. Como podrás observar la densidad y el tamaño de las pantallas tienen una relación que puede una afectar a la otra para mayor o para menor calidad, esto es un tema que debemos de conocer ya que al empezar a crear interfaces para nuestras aplicaciones tendremos que manejar el tema de la calidad y tamaño de los elementos.

© Alfaomega - RC Libros

10 MEDIDAS EN ANDROID

En este capítulo veremos las diferentes medidas que tenemos en Android para asignar valores de tamaño o espacio a los elementos y explicaremos en qué consiste cada una de ellas y cuándo es adecuado utilizarlas.

Comenzaremos nombrando cada una de ellas:

- DP (Density –independent Pixels)
- PT (Points)
- PX (Pixels)
- MM (Milimeters)
- IN (inch)
- SP (Scale independent Pixels)

Cada una de estas medidas tiene usos diferentes dependiendo de la situación o del elemento que lo requiera. Así como en la vida tenemos los centímetros y las pulgadas, aquí tendremos diferentes unidades de medición.

Píxeles (PX)

Comenzaremos con el más conocido, no importa si trabajaste en web o con aplicaciones de escritorio o si eres diseñador, el píxel es la medida universal para la representación gráfica en la informática, lamentablemente esta medida no sirve como referencia en el mundo Android. Recordemos que el pixel se considera la unidad mínima de una pantalla, por eso las pantallas se miden de las siguientes formas:

- 1024 x 800
- 1210 x 900

Lo que quiere decir esto es que contamos con 1024 píxeles por un lado y 800 por otro lado, así que nos podemos hacer una idea de cómo es el tamaño de la pantalla y podemos deducir con relación a este número que es una pantalla en forma de rectángulo, más ancha que alta. Aunque recordemos que la medida más importante en las pantallas es la línea diagonal y esta se mide en pulgadas.

El píxel es una medida física y por lo tanto inmutable, es decir, no varía, este mismo punto es el que hace que esta referencia no sea buena para Android ya que en Android podríamos tener 2 pantallas una muy grande y una más pequeña y no significa que la grande sea mejor que la pequeña, esto se debe a otros conceptos como el de densidad, que afecta a esta percepción, por lo tanto el píxel es una medida que se utiliza en el momento de crear una imagen o recurso visual, pero para finalmente asignar un valor en Android debemos de pensar en las medidas que veremos más adelante.

En resumen: olvídate de los píxeles en Android, está bien que el diseñador siga utilizándolos cuando trabaja en Photoshop, pero simplemente es inútil en Android, en otras palabras, nunca debes de escribir dentro del código que tu medida de referencia es el píxel o tendrás problemas en el momento de ver tu aplicación.

Pulgada (inch)

Las pulgadas no tienen mucho misterio, siempre serán pulgadas, que es equivalente a 2,54 cm y sucede lo mismo que con los píxeles en Android, se utilizan en casi nada ya que para eso tendremos las medidas propias de Android, así que evita lo más que puedas estas dos medidas físicas, tendrás problemas si las utilizas como base en tus proyectos.

Milímetros (mm)

En el caso de los milímetros sucede lo mismo que con la pulgada y los píxeles, somos conscientes de que existen, pero simplemente no tendríamos por qué usarlos salvo casos muy especiales, por lo tanto, si podemos evitarlos, manejaremos las cosas de forma correcta.

© Alfaomega - RC Libros

Puntos (pt)

Los puntos son matemáticamente 1/72 de una pulgada, solamente debemos de tomar en cuenta el tamaño de la pantalla.

Píxeles de densidad independiente (dp)

En la documentación oficial los encontraremos como una unidad abstracta, eso se debe a que físicamente no los podemos medir con una regla o con cualquier dispositivo, esta unidad habrá que calcularla.

Un pixel en dp es equivalente a un píxel físico en una pantalla de 160 DPI.

Para continuar te explicaré qué es DPI.

DPI o en español los puntos por pulgada son la cantidad de píxeles que existen en una pulgada (DENSIDAD) algo que mencionamos en el capítulo anterior al hablar de densidad, había un valor numérico que se considera para referirse a la densidad. Cuantos más puntos pixeles tengamos en una pulgada, mejor imagen tendremos ya que podremos dibujar más claro y tener una gama más amplia para dibujar el elemento.

En valores numéricos cada densidad representaría la siguiente equivalencia:

- ldpi 120 dpi
- mdpi 160 dpi
- hdpi 240 dpi
- xhdpi 320 dpi
- xxhdpi 480 dpi
- xxxhdpi 640 dpi

Volviendo al concepto de dp, retomemos que la equivalencia de 1 dp sería lo mismo que un pixel físico en una pantalla de 160 dpi.

Realizaremos un ejemplo sencillo, un elemento que queremos dibujar tiene el tamaño de 2 dp, supongamos que la pantalla que tenemos en este dispositivo es de 160 dpi, por lo tanto, el cálculo de pixeles sería también 2. Así que si queremos dibujar un elemento con 2 dp de ancho y 2 dp de alto sería lo mismo a tener uno con 2 píxeles de ancho y 2 píxeles de alto.

Ahora queremos hacer lo mismo pero para una pantalla XHDPI, es decir, de 320 dpi, si hacemos la misma operación con la fórmula sería el doble, esto es, si queremos colocar un elemento con 2 dp de ancho x 2 dp de alto, en una pantalla de 320 dpi necesitaríamos un elemento con 4 píxeles de ancho y 4 de alto, es decir, que sería el doble que el cálculo que hicimos con dpi, pero visualmente cuando el usuario abre la aplicación en un teléfono con dpi y con hdpi va a ver el elemento abarcando la misma proporción y todo es gracias a que hicimos el cálculo de la manera correcta. A continuación, se mostrará una tabla con la escala de las imágenes y un ejemplo en medidas.

Resolución	Dpi	Escala	Tamaño imagen ejemplo
mdpi	160	1.0	100x100
hdpi	240	1.5	150x150
xhdpi	320	2.0	200x200
xxhdpi	480	3.0	300x300
xxxhdpi	640	4.0	400x400

© Alfaomega - RC Libros

11 PANTALLAS PARA MÚLTIPLES DISPOSITIVOS

Para crear una pantalla en Android debemos de tener en consideración varios factores, en este capitulo abordaremos los factores que pueden marcar la diferencia de hacer una aplicación con calidad o hacer una aplicación por hacer.

La mayoria de las aplicaciones Android no respetan estas pautas de calidad que nos proporciona y nos sugiere el mismo Android en la documentación oficial, esto se debe al gran trabajo que esto conlleva cuando tenemos que hacer una aplicación, además de que en muchos casos se desconocen las buenas pautas de diseño de pantallas e interacción en Android.

Comenzaremos con el tema de la orientación de pantalla, ya que es uno de los temas con más cuestionamientos por la naturaleza de su funcionamiento, ya que en el caso de Android se maneja de una forma única en comparación con los otros sistemas operativos.

Orientación de pantalla

En el sistema operativo Android tenemos dos orientaciones de pantallas el teléfono lo podemos colocar en dos posiciones, en la posición vertical y en la posición horizontal. Esto tiene su nombre técnicamente, cuando el teléfono está en forma vertical decimos que se encuentra en estado PORTRAIT y cuando en estado horizontal está en estado LANDSCAPE.

Cada uno de los estados es independiente y desde ahora debemos de tener claro que no existe el mejor o peor estado, aunque existen casos en donde el usuario puede estar más cómodo dependiendo de la naturaleza de la aplicación, por ejemplo la mayoria de los juegos en Android optan por una orientación horizontal ya que esta posición le proporciona al usuario una mayor visión del escenario, pero existen

juegos en los que el usuario puede seleccionar entre horizontal y vertical, así que queda a gusto del usuario tomar la decisión.

En el caso de las aplicaciones es muy importante entender que la usabilidad se ve afectada dependiendo de la posición, ya que por naturaleza Android lo que hace es abarcar el ancho que le fue indicado o el alto, así que en la mayoría de los casos la mejor idea es crear una interfaz para que el usuario se sienta cómodo en una posición o en otra, pero además obtenga una buena experiencia de cualquiera de las dos.

La regla entonces es crear una interfaz para cada una de las vistas, portrait y landscape.

Portrait

También conocida como la vista vertical con la que habitualmente se usa el teléfono, esta vista es la que por defecto tenemos en todas las aplicaciones, así que para generar un layout que respete esta vista simplemente tenemos que colocar dentro del directorio res/layout/ un archivo xml, si tomamos como base el ejemplo de "hola mundo" tendríamos una estructura como la de la siguiente imagen:



© Alfaomega - RC Libros

Podemos observar que el "Hola mundo" ya tenía entonces como decíamos por efecto una pantalla considerada portrait. Este layout contiene este código:

```
<?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"
   android:padding="50dp"

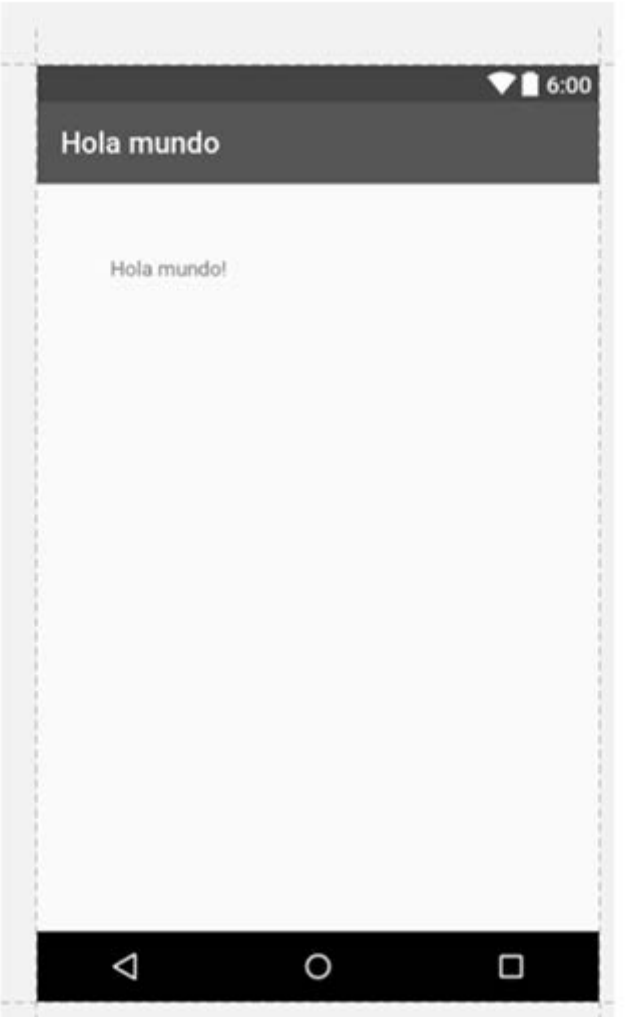
tools:context="studio.android.libro.josedlujan.com.holamundo.MainActivity">

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

</LinearLayout>
```

Aunque el código no tiene mucha importancia, lo mostramos para que el lector vea que por ahora no tenemos nada especial, simplemente crear un layout es la forma de crear por defecto una vista Portrait.

La aplicación se ve así:



© Alfaomega - RC Libros

Landscape

La vista landscape o vista horizontal es la vista que puede ayudar de manera interesante a un usuario a disfrutar de la comodidad si está bien aplicada, por lo mismo si nuestra aplicación quiere tener una ventaja de calidad y un sello de profesionalismo, debería de contener esta vista en caso de ser necesario.

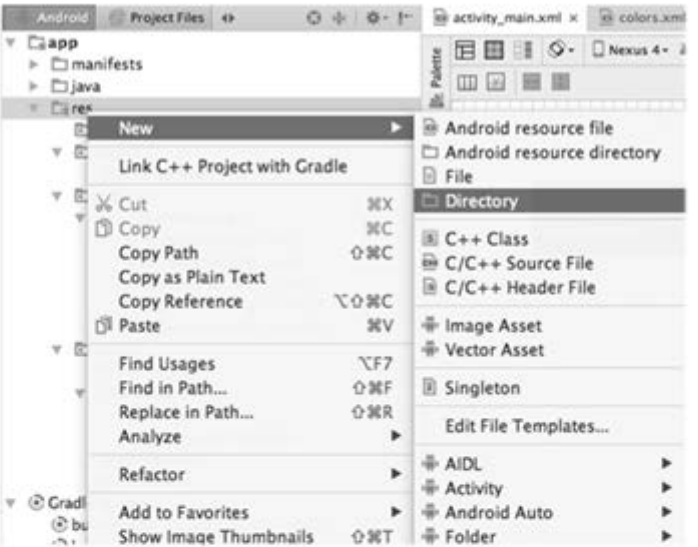
Para crear una vista horizontal debemos crear un directorio con el mismo nombre, el nombre "layout" y agregar la palabra "-land", las palabras deben ir separadas por un guion, con esa creación del directorio ya sabe Android que tiene un directorio en donde va a encontrar archivos xml layouts listos para usarse en el modo horizontal, pero además de eso deben de tener el mismo nombre, así que si tenemos un archivo que se llama activity_main.xml en el directorio layout, en el directorio layout-land debe de tener otro archivo con el mismo nombre, Android por sí solo ya sabrá que cuando el usuario hace la rotación del teléfono debe de mostrar la otra activity declarada.

Es importante recalcar que las actividades no están obligadas a tener los mismos elementos, por lo tanto, si estas actividades tienen diferentes elementos no pasa nada, aunque el desarrollador o la empresa que hace la aplicación deberá de preocuparse por la experiencia de usuario y que esta no se vea afectada por la falta de elementos en una y otra pantalla.

Vamos a seleccionar el directorio "res" y hacemos clic derecho, vamos a seguir este camino:

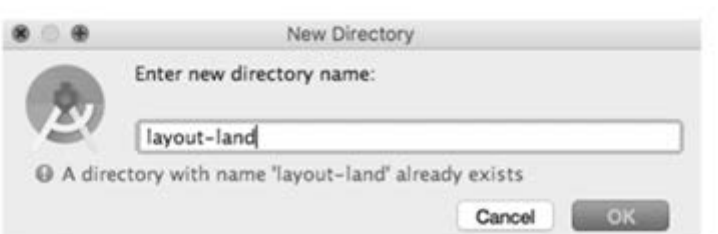
```
res -> New -> Directory
```

Así se ve en Android Studio:



Le colocamos el nombre de "layout-land" al directorio y listo. Hemos creado el directorio.

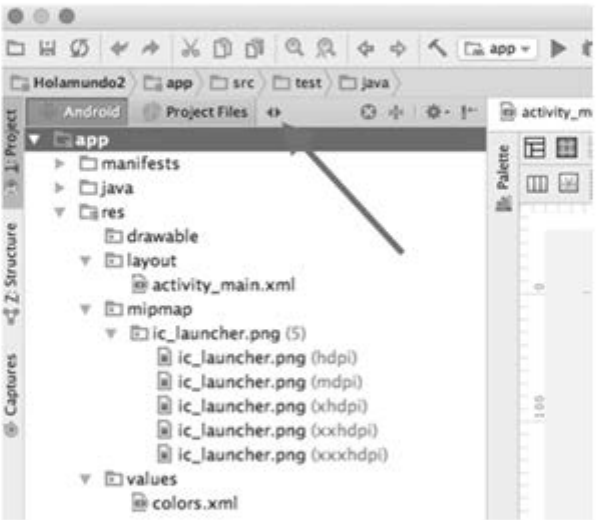
Si observamos los directorios veremos que el directorio en realidad no aparece o por lo menos no se ve, pero si vuelves a intentar el procedimiento, te mostrará un mensaje como el siguiente:



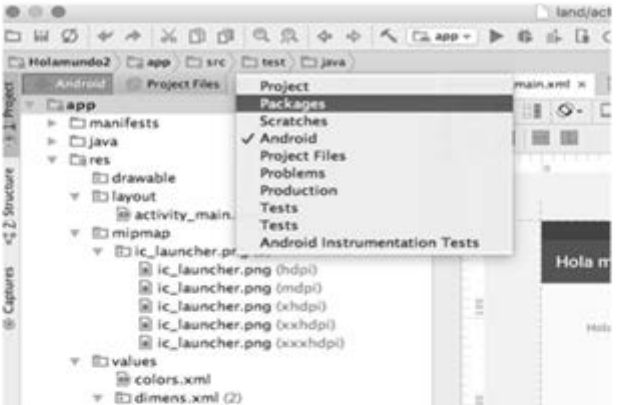
© Alfaomega - RC Libros

Nos avisa de que, aunque no lo veamos, el directorio sí fue creado, esto no se ve debido a la perspectiva, así que lo que tendremos que hacer es cambiar la perspectiva con la que estamos viendo la estructura del proyecto por otra, en este caso la perspectiva que nos viene bien utilizar para encontrar el directorio es la "Package".

Cambiaremos la perspectiva que tenemos de Android por la perspectiva "package" como se ve en la imagen, pulsamos en el botón con los dos triángulos:



Después vemos el menú que nos muestra y hacemos clic en la opción "package":



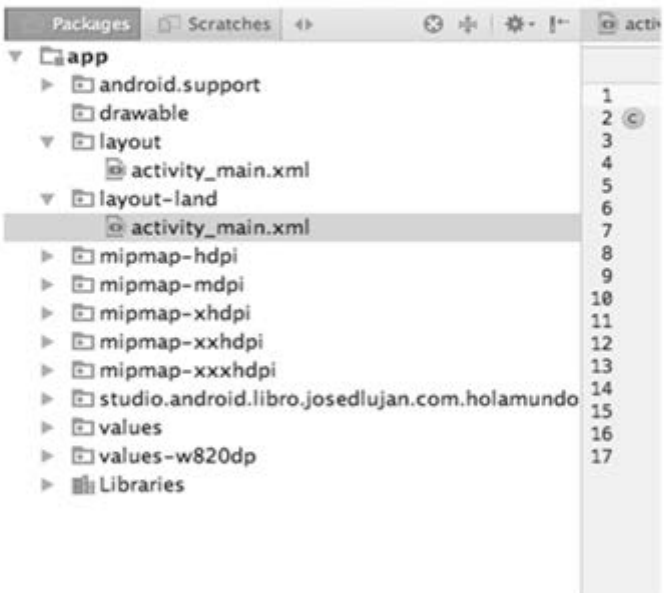
Ahora veremos una estructura diferente que será la siguiente:



© Alfaomega - RC Libros

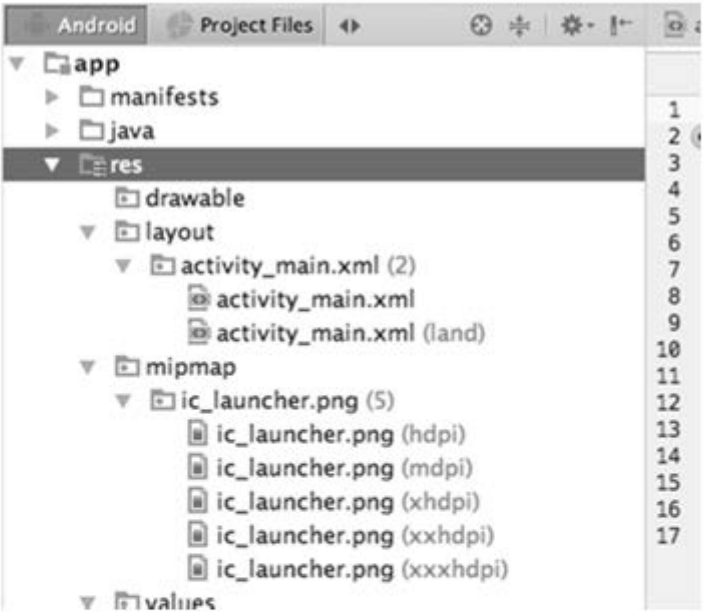
Como vemos en la imagen anterior tenemos el directorio layout y debajo el directorio layout-land, así que hemos creado el directorio en donde vamos a colocar las interfaces landscape para que Android las tome de ahí.

Algo que podemos hacer por ahora es copiar el archivo que ya tenemos como activity dentro del directorio layout y pegarlo en el directorio layout-land, debemos de colocarle el mismo nombre, así que no es problema copiar y pegarlo. Después de copiar y pegarlo tendremos la estructura como la de la siguiente imagen:



Es muy importante recalcar que el nombre debe ser idéntico ya que en caso contrario no funcionaría lo que intentamos, que es colocar la interfaz PORTRAIT y LANDSCAPE.

Ahora regresaremos a la perspectiva Android y veremos la estructura del proyecto, tendremos algo parecido a la siguiente imagen:



Como vemos el directorio res contiene un directorio layout, pero este directorio contiene dos archivos activity_main y uno tiene entre paréntesis la palabra land para indicar que este es el archivo que se utilizará para la vista landscape.

Para hacer una prueba de la diferencia de vistas vamos a cambiar el tamaño de la fuente y el texto del textview de la activity que es para landscape.

El archivo main_activity.xml de PORTRAIT tendrá este código:

```
<?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"
```


© Alfaomega - RC Libros

```
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="50dp"

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

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

</LinearLayout>
```

Es el "Hola mundo" sin nada especial. Ahora vamos a cambiarle la fuente al archivo y el texto del textview; 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"
    android:padding="50dp"

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

    <TextView
        android:text="Hola mundo en Landscape!"
        android:textSize="40sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

© Alfaomega - RC Libros

Las únicas dos líneas que agregamos son estas:

```
android:text="Hola mundo en Landscape!"
android:textSize="40sp"
```

Así que ahora vamos a probar la aplicación en el emulador:



Ahora podemos hacer girar el emulador con el botón que tiene el menú del emulador a la derecha, lo señalamos en la siguiente imagen:

© Alfaomega - RC Libros



Al girar el emulador veremos el cambio de forma automática sin que nosotros hagamos algo extra.



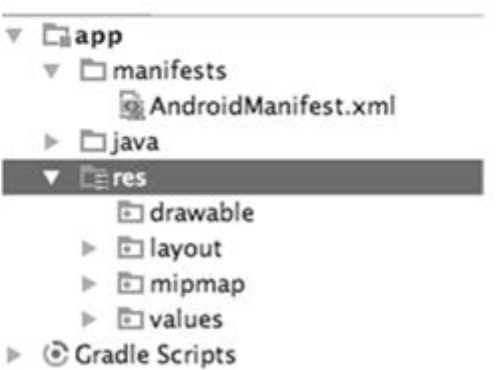
Así es como Android detecta de forma automática cuál es el layout que va a mostrar, en realidad depende de que nosotros creemos los layouts correspondientes para los diferentes tipos de pantallas que podemos tener.

Forzar solamente una opción de orientación

Nos podremos encontrar por alguna situación en la necesidad de solamente permitir al usuario el uso de solo un tipo de vista, o portrait o landscape, por ejemplo, en los juegos muchos te hacen usar desde el inicio la vista horizontal, ya que es como tienen planeado que el juego funcione.

Para lograr esto tenemos que forzar la acción, pero es importante mencionar que Google, la empresa que se encuentra detrás de Android, nos comenta en la documentación oficial que esto no es lo ideal, que debemos siempre crear una pantalla para cada orientación, así que, si es por pereza o por ahorrar trabajo es algo bastante malo, pero si es por el bien del usuario y porque es la mejor forma de utilizar la aplicación, adelante.

Vamos a tener que ir al archivo Manifest, que se encuentra en la parte superior de nuestros directorios en la aplicación:



© Alfaomega - RC Libros

Si abrimos el archivo androidManifest.xml vemos 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>
```

Debemos de ubicar la actividad a la que queremos forzar y que esta no se pueda ver en las dos orientaciones y debemos de colocar el siguiente código:

```
<activity    android:screenOrientation="portrait"
android:configChanges="orientation|keyboardHidden">
 </activity>
```

Con estas instrucciones en cada una de las actividades que necesitamos se mantengan siempre en una orientación, lograremos forzar este comportamiento.

Si tenemos 20 actividades y queremos que las 20 mantengan la pantalla vertical, tendríamos que aplicar este código para cada una de las actividades. La línea que indica la orientación es la primera:

```
android:screenOrientation="portrait"
```

En la segunda línea nosotros le estamos indicando a Android que nos hacemos responsables de la orientación y no él, además de ocultar el teclado:

```
android:configChanges="orientation|keyboardHidden">
```

Si agregamos estas líneas a la única activity que tenemos en nuestra aplicación, quedaría de la siguiente forma:

```
<?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"
               android:screenOrientation="portrait"
               android:configChanges="orientation|keyboardHidden" />

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

</manifest>
```

Tamaños de pantallas

Ya se ha mencionado con anterioridad en el libro que tenemos 4 tamaños de pantalla:

© Alfaomega - RC Libros

- Pequeño
- Normal
- Largo
- Extra Largo

Estos cuatro tamaños los tenemos que considerar también al hacer una aplicación. Esta es la razón por la que vamos a tener que crear diferentes layouts dependiendo de los tamaños de pantalla.

Para crear los layouts hacemos clic con el botón derecho al directorio res y vamos a crear el directorio layout-large:



Ahora colocamos el nombre del directorio layout-large:



Si nos vamos a la vista de project, podemos ver que ya tenemos el nuevo directorio para crear un layout correspondiente a una pantalla larga.



Por ahora el directorio se encuentra vacío, lo que haremos es copiar y pegar el activity_main.xml del directorio layout, después de eso ya deberíamos de ver un archivo activity_main.xml dentro de los directorios layout, layout-land y layout-large:



© Alfaomega - RC Libros

Los dispositivos que cuenten con una pantalla grande ahora también podrán tener una pantalla personalizada para las dimensiones "grandes", por ejemplo, si queremos tener también en cuenta la pantalla extra larga, tenemos que crear un directorio que es layout-xlarge. Repetiríamos el mismo procedimiento solo que con el nombre nuevo y tendríamos el directorio de archivos como en la imagen siguiente:



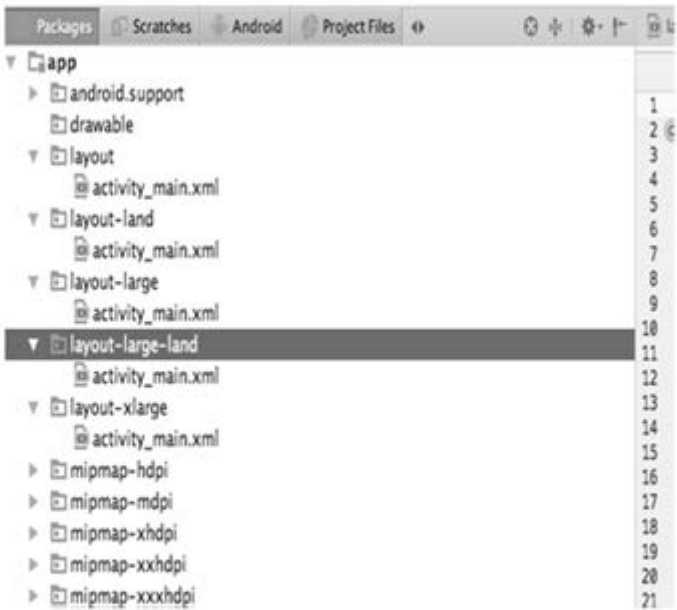
Por ahora tendremos en cuenta las pantallas:

- Layout – Directorio pantalla normal en vista portrait
- Layout-land – Directorio pantalla normal en vista landscape
- Layout-large – Directorio pantalla larga con vista portrait
- Layout-xlarge – Directorio pantalla extra larga con vista portrait

Si ponemos atención notaremos que solamente tenemos considerada una vez la pantalla en posición landscape, así que deberíamos de crear una interfaz para pantalla larga o extra larga, pero en posición Landscape.

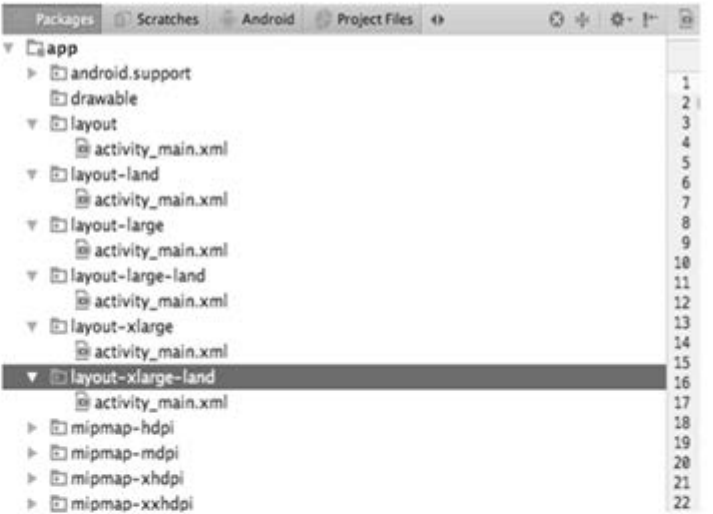
© Alfaomega - RC Libros

Agregaremos primero el directorio layout-large-land dentro de nuestra estructura, colocaremos de igual forma el archivo activity_main.xml solamente para colocarle algo, el lector ya debería de conocer que a estas alturas el archivo xml lo podemos personalizar como queramos, así que no importa eso, pero algo muy importante es que siempre deben de llevar el mismo nombre:



Agregaremos después el segundo layout, layour-xlarge-land, también contará con el archivo que tenemos como base para las actividades en este "Hola mundo":

© Alfaomega - RC Libros



Podemos decir que estamos contemplando todos los tamaños de pantallas que tenemos disponibles, el tamaño pequeño de pantalla hoy en día es casi un mito, ya que ningún fabricante crea dispositivos con el tamaño mínimo, el precio de la tecnología ha bajado y este es un factor a considerar por qué los fabricantes ya no se preocupan por eso.

12 ARCHIVO STRING

El archivo string tiene un uso muy importante dentro de las aplicaciones, no solamente nos sirve como un apoyo en el momento de colocar texto en una aplicación, la labor de este directorio y de directorios similares es mucho más compleja en comparación con lo que se cree de estos.

Si queremos crear una aplicación para diferentes países o idiomas debemos de conocer a detalle el archivo string. Este archivo nos apoya con el idioma dentro del mundo Android.

Lo que tenemos que hacer es crear el directorio adecuado para el idioma en el que vamos a escribir nuestra aplicación. No tenemos límite de idiomas en los que podemos dejar disponible nuestra aplicación. Lo importante es crear la estructura de archivos adecuada, vamos a ir al directorio:

```
res/values/strings.xml
```

Abriremos el archivo y tendremos el siguiente código:

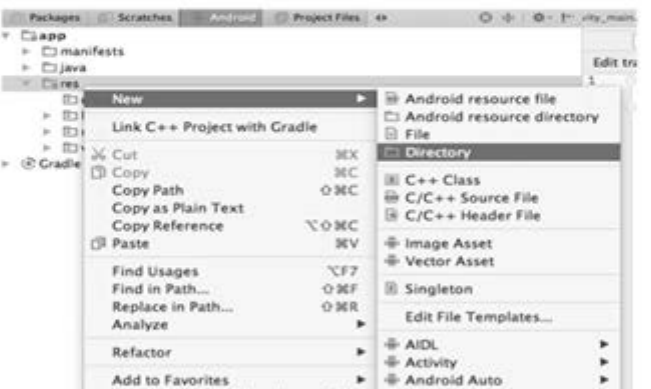
```
<resources>
    <string name="app_name">Hola mundo</string>
</resources>
```

Por defecto solo contamos con la cadena "app_name" que siempre va a coincidir en un principio con el nombre de la aplicación, es el que colocamos en el momento de crear el proyecto. Esta cadena de texto puede ser editada y cambiada en cualquier momento.

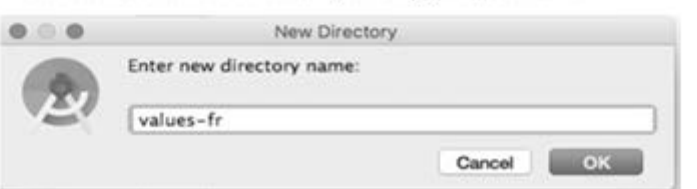
Si cambiamos el texto de esta cadena, en toda la aplicación cuando se mande a llamar a la cadena "app_name" también se va a notar el cambio. Podemos ver que esta forma de trabajar también es muy cómoda ya que una actualización a este archivo puede tener efecto en toda la aplicación que estamos desarrollando.

Si pensamos en una aplicación que queremos que también esté disponible también para la gente que habla francés, lo que tenemos que hacer es crear otro archivo de cadenas, pero exclusivamente en francés, esto equivale a tener 2 archivos con las mismas cadenas de texto, es decir, que tengan el mismo nombre, pero que además tengan dentro de cada archivo las cadenas en el idioma correspondiente.

Primero vamos a crear el directorio para una aplicación con el idioma francés soportado. Vamos a hacer clic con el botón derecho en el directorio res y le vamos a dar a crear un nuevo directorio:



Le colocaremos el nombre de values-fr, en la imagen lo podemos ver:



© Alfaomega - RC Libros

Ahora vamos a ver cómo en nuestra vista de directorios hay un nuevo directorio:

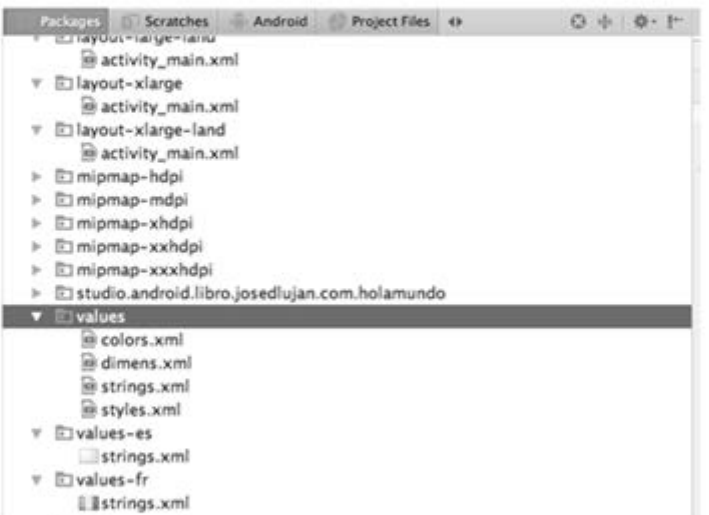


Por ahora podemos copiar y pegar el archivo strings.xml, también podemos crear el archivo desde cero, pero para tener una mayor velocidad copiamos y pegamos. La estructura quedará de la siguiente forma:



Vemos 2 directorios con la palabra values, el detalle importante es que el directorio que contará con un archivo de cadenas de texto en francés tiene como icono una bandera de Francia, esto lo sabe Android ya que el directorio termina con "-fr", esto le indica que las cadenas de texto que contendrá el archivo strings.xml se van a tomar cuando el usuario tenga en su teléfono como idioma predeterminado el francés.

Ahora vamos a agregar otro directorio values-es, repetimos el procedimiento y veremos algo parecido a lo siguiente:



Si observamos el icono, ahora sale de color blanco, pero no afecta al funcionamiento de nuestra aplicación, encontraremos casos en donde suceda esto.

Al echar un vistazo rápido a la vista "Android" deberíamos de ver algo así:

© Alfaomega - RC Libros



Podemos ver que en esta vista vemos solamente un directorio strings.xml y dentro de este vemos 3 directorios y al final 2 de ellos cuentan con paréntesis y el idioma que contiene uno es "es" de español y el otro es "fr" de francés.

Si abrimos los archivos y queremos que la aplicación se ejecute de manera correcta, debemos de tener lo siguiente:

Archivo values/strings.xml

```
<resources>
   <string name="app_name">Hello wolrd!</string>
</resources>
```

Archivo values-es/strings.xml

```
<resources>
   <string name="app_name">Hola mundo! </string>
</resources>
```

Archivo values-fr/strings.xml

```
<resources>
   <string name="app_name">Salut monde!</string>
</resources>
```

Podemos observar que todos los archivos contienen la misma cadena de texto que es "app_name" pero ya vemos el texto que cada uno incluye la frase "Hola mundo" en su idioma, por lo tanto, estamos creando una aplicación multi-idioma, podrá soportar en este caso 3 idiomas, inglés, francés y español.

En caso de que nosotros no le indiquemos ningún idioma, simplemente tomará el archivo strings.xml que contiene en el directorio principal que es values.

Nosotros no tenemos que hacer nada más, Android por sí solo sabe qué idioma debe de desplegar, esto se debe a que el usuario cuando compró su teléfono en la configuración colocó un idioma por defecto, siempre los teléfonos ya vienen con una pequeña preconfiguración también en la que se le asigna el idioma dependiendo el país, en este caso es natural que venga con español.

Si quieres probar que el "Hola mundo" aparezca en francés, lo único que tienes que hacer es entrar en la configuración de tu teléfono o emulador e indicarle que el idioma por defecto es este, con eso ya al abrir la aplicación, el "Hola mundo" te saldrá en el idioma seleccionado si es que existe, en caso contrario saldrá el que tenemos por defecto, que en este caso es el inglés.

© Alfaomega - RC Libros

13 IMÁGENES

En este capítulo abordaremos las imágenes y conoceremos algunas reglas que tenemos que respetar cuando queremos trabajar con ellas.

El elemento más común para colocar imágenes dentro de una aplicación en Android es el elemento ImageView, este elemento nos permite definir un contenedor al cual le podemos asignar una imagen de las que ya tenemos dentro de nuestra aplicación.

Características de imágenes

Una de las cosas más importantes dentro del manejo de imágenes en el mundo del desarrollo Android es que se recomienda que las imágenes en su mayoría sean en formato PNG, la principal razón es la transparencia que vamos a necesitar en muchos casos para que estas no tapen el fondo de pantalla y además sea más fácil realizar el diseño de la aplicación. Si contamos con un diseñador en el equipo del desarrollo de la aplicación, lo mejor que le podemos decir es que las imágenes nos las entregue en formato PNG.

El formato JPG no está excluido, pero al tener que colocar un fondo en la imagen con un color, nos limita el uso ya que muchos elementos van a interactuar entre ellos y un fondo en la imagen es una desventaja para el acomodo de imágenes.

Las imágenes deben de colocarse siempre en un directorio, en este caso:

drawable

Este directorio es donde por defecto colocaremos las imágenes que se utilizarán en nuestra aplicación, desde ahí traeremos las imágenes a nuestro contenedor ImageView.

ImageView

El imageview como cualquier otro elemento visual en Android necesita tener al menos 2 propiedades, el alto y el ancho, aunque por su naturaleza de ser un contenedor necesitamos siempre colocarle la fuente, esta propiedad lo que hace es indicarle la ubicación donde se encuentra la imagen que le corresponde. Vamos a colocar un elemento ImageView dentro de un linearlayout, este sería el código:

```
<?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">


    <ImageView
        android:src=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </ImageView>


</LinearLayout>
```

El bloque LinearLayout lo podemos ver como cualquier bloque que ya estudiamos anteriormente.

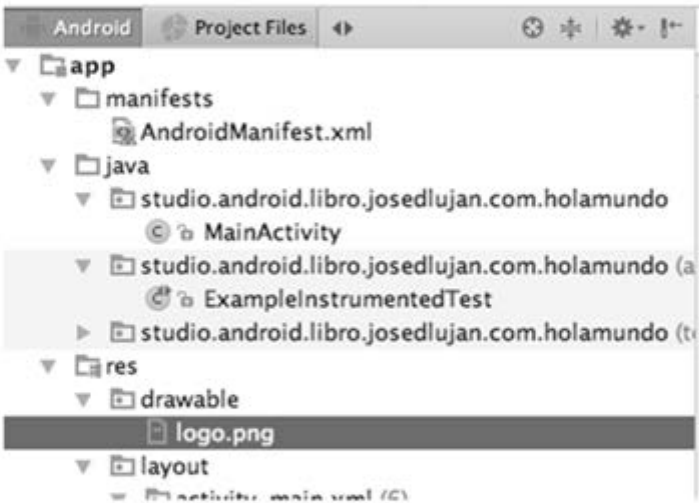
El bloque del ImageView contiene el ancho y el alto, la propiedad src="" es la que nos indica la fuente de donde tomará la imagen. Por ahora se encuentra vacía, pero colocaremos una imagen dentro de Android Studio.

Tenemos varias opciones para colocar una imagen, veamos las 2 más comunes:

© Alfaomega - RC Libros

- Arrastrar la imagen y soltarla dentro del directorio drawable (DRAG & DROP).
- Copiar con "ctrl + c" desde la ubicación en nuestra computadora y pegar con "crtl + v" en el directorio drawable.

Cualesquiera de las dos opciones anteriores nos van a servir para nuestro cometido. Ya que hemos copiado la imagen la vamos a ver de la siguiente forma en el directorio, la imagen tiene el nombre de logo.png:



Podemos ver que en el directorio drawable ya tenemos nuestro primer archivo. Ahora vamos a insertar esta imagen en el ImageView en el atributo src. El código queda de la siguiente forma:

```
<?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">

    <ImageView
        android:src="@drawable/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </ImageView>

</LinearLayout>
```

Vemos que solamente cambió la línea de la fuente:

```
android:src="@drawable/logo"
```

Con esa línea le indicamos la fuente de la imagen que va a contener.

Directorios drawable

Por ahora como se ve en la estructura de nuestros archivos tenemos un directorio drawable, pero en realidad podemos tener más directorios dependiendo de las necesidades de nuestra aplicación. Esto tiene relación con el tema de densidades que ya vimos anteriormente.

Si no hacemos bien nuestro trabajo esto se verá como la siguiente imagen:



Esta imagen es la explicación visual que nos proporciona la documentación de Google cuando hablamos de densidades no resultantes del modo correcto. Cuanto

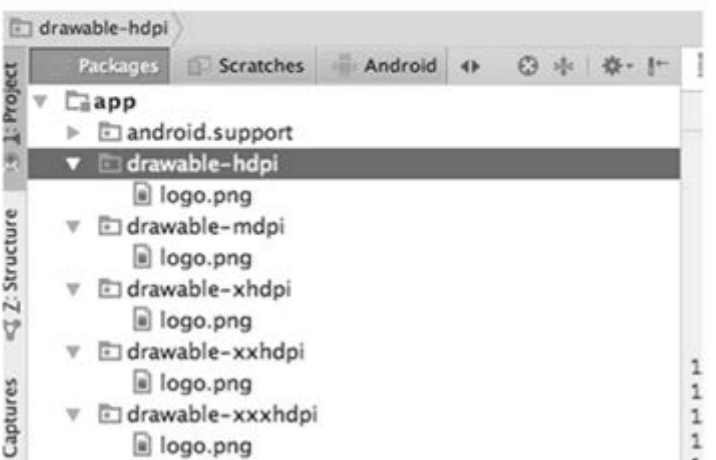
© Alfaomega - RC Libros

más alta es la densidad o más baja es la densidad, más se va a distorsionar el tamaño de la imagen, esto es debido a la densidad y la naturaleza de los puntos.

Lo que tenemos que hacer para evitarlo es crear los directorios correspondientes para almacenar las imágenes que vamos a usar. En nuestro caso vamos a considerar las siguientes densidades:

- Media
- Alta
- Extra alta
- Extra extra alta
- Extra extra extra alta

Tendremos que crear esos directorios. A estas alturas del libro ya sabemos hacerlo, simplemente compartiré en una imagen cómo debe de quedar la estructura:



La perspectiva de la imagen anterior es la de Packages, ahora vamos a ver cómo lo observaremos en la vista habitual "Android":



Vemos que el directorio drawable contiene un directorio logo.png y dentro de este tenemos 5 archivos logo.png y al final de estos entre paréntesis tiene la referencia a la densidad que pertenece:

- hdpi
- mdpi
- xhdpi
- xxhdpi
- xxxhdpi

© Alfaomega - RC Libros

Ya tenemos la primera parte que es la del directorio, ahora debemos de calcular el tamaño de las imágenes para cada una de las densidades, este es el trabajo habitual del diseñador, pero en la mayoría de los casos es algo que el diseñador desconoce, ya que viene del mundo del desarrollo web o de otras formas de diseño, es por eso que Android cuenta con temas como este que lo hace peculiar o un poco complejo para el diseñador tradicional.

Escala de imágenes

Existe un número mágico en Android para poder crear las imágenes y que estas se vean de manera correcta en todos los dispositivos. Tenemos que tener en cuenta el tema de las densidades y el cálculo. El número es:

3:4:6:8:12:16

Este número nos va ayudar a crear las imágenes de las medidas correctas para todas las escalas.

Paso 1. Selección tamaño base

Primero vamos a crear la imagen del tamaño más grande, se recomienda utilizar como base las densidades más grandes o pantallas más grandes para afectar lo menos posible la calidad de la imagen, en el caso de ser un vector o una imagen vectorizada no habrá mucho problema, pero con otros formatos esto puede complicar el procedimiento, por lo tanto el diseño base del que vamos a partir para la generación de imágenes dependiendo de la escala es del tamaño más grande que podemos tener, así evitamos problemas de calidad de imagen.

Partamos entonces de que tenemos una imagen de 192x192, sería lo mismo que tener una de 200x250, 400x500, etc. Lo importante es entender lo que significa la escala y qué es lo que esto implica. La única recomendación que por facilidad se aconseja seguir es la de que las medidas que se recomiendan estén en números pares, por ejemplo 64, 36, 192, 180, si se utiliza un número impar, en el momento de generar la escala puede causar confusión a alguien que está iniciándose debido a los números fraccionados.

Entonces seleccionamos como tamaño base el 192x192 para nuestra imagen.

Paso 2. Aplicar la escala

El segundo paso que vamos a aplicar es calcular los tamaños de nuestras imágenes, por lo tanto, lo que haremos es utilizar el número que ya vimos anteriormente:

3:4:6:8:12:16

Debemos de considerar que esta es la escala hoy en día, pero esto puede variar en algunos meses, en un principio no teníamos tantas densidades de pantallas, por lo tanto, cuantas más densidades tengamos más valores tendremos en la escala.

Veamos cómo funcionaría entonces con el valor 192. Este número es el mayor, por lo tanto sería el equivalente al número 16. Es como si 192 fuera nuestro 100%.

Entonces 192 = 16. 192 es 16 veces el valor. Por lo tanto, dividiremos este número 192/16 y obtendremos el valor de 12.

12 será el número mágico ya que con este calcularemos todas las escalas, retomando el número:

3:4:6:8:12:16

Lo que haremos es calcular el tamaño para cada una de las escalas:

Escala 3 será igual a 12 * 3 = 36, por lo tanto la imagen será de 36 x 36.

Escala 4 será igual a 12 * 4 = 48, por lo tanto la imagen será de 48 x 48.

Escala 6 será igual a 12 x 6 = 72 por lo tanto la imagen será de 72 x72.

Escala 8 será igual a 12 x 8 = 96, por lo tanto la imagen será de 96 x 96.

Escala 12 será igual a 12 x 12 = 144, por lo tanto la imagen será de 144 x 144.

Escala 16 será igual a 12 x 16 = 192, por lo tanto la imagen sería de 192 x 192, como vemos con este último cálculo se corrobora que sí hicimos el cálculo de la escala bien, y ahora ya sabemos el tamaño de todas las imágenes que tendremos que hacer para el funcionamiento correcto de nuestra aplicación.

© Alfaomega - RC Libros

Pase 3. Hacer las imágenes

En esta última etapa en realidad teóricamente el desarrollador no hace nada, pero depende del flujo de trabajo que tenga la aplicación o el equipo de desarrollo, si solamente una persona está desarrollando la aplicación, entonces esa persona tendrá que hacer las 3 etapas, en una empresa de desarrollo de aplicaciones tendría entonces que comenzar a trabajar el diseñador para entregar las imágenes en cada uno de los tamaños requeridos.

No siempre es generar todas las imágenes para todas las escalas, en alguna ocasión por necesidad, por falta de interés o por no contar con presupuesto se puede decidir solo trabajar para algunas densidades y no para todas, no es nada extraordinario, simplemente solo se crean los directorios necesarios que ya se mencionaron anteriormente y se colocan las imágenes correspondientes como se mencionó en el paso 2.

No es obligatorio crear una aplicación para todas las densidades, pero si es recomendable ya que el usuario gozará entonces de una buena calidad de aplicación y podrá tener una experiencia natural en el uso de esta.

Para la creación de imágenes, la mayoría de los desarrolladores hoy en día utilizan Photoshop, pero en realidad no existe algo especial que haga que seleccionen esta herramienta, es más por comodidad del diseñador o por conocimiento de la herramienta, pero si se puede generar un formato PNG, es más que suficiente para la creación de las imágenes.

© Alfaomega - RC Libros

14 INTERACCIÓN EN UNA APLICACIÓN

En este capítulo nos vamos a centrar en el código Java dentro de una aplicación y además conoceremos cómo sucede la interacción de los elementos dentro de una aplicación. Comenzaremos abriendo el archivo MainActivity.java que tenemos creado en nuestra aplicación "Hola mundo" y explicaremos sus partes. El archivo contiene este código:

```
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);
    }
}
```

En el primer apartado como en cualquier archivo Java de esta aplicación y de cualquier otra contiene el paquete al que pertenece:

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

Ahora vemos los dos imports que estará utilizando, AppCompartActivity y Bundle.

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

Finalmente comienza la declaración de la clase MainActivity que extiende de la clase AppCompartActivity.

```
public class MainActivity extends AppCompatActivity {
```

Java, Programación orientada a objetos

Este libro no intenta ser una guía de programación o una guía para usuarios que tratan de aprender a programar, pero sí es importante que expliquemos unos puntos de Java que son esenciales para entender toda la información escrita en este libro. Algunos de los conceptos importantes son:

- Objeto
- Clase
- Herencia
- Método

Hay más conceptos, pero estos 4 son necesarios para que el desarrollador Android por muy novato que sea los entienda desde un inicio y así facilitarle el conocimiento del código en el desarrollo de aplicaciones y por lo tanto de este libro, los demás conceptos avanzados de la programación orientada a objetos corren por cuenta del lector después de terminar el libro, ya que como se mencionó, el libro está centrado en el desarrollo de aplicaciones móviles, no en el aprendizaje de programación.

Clase

Si existe una palabra relacionada con Java desde sus inicios es la palabra clase, en otros lenguajes de programación se puede comenzar por otros lados a aprender a programar, en el caso de Java no se puede omitir desde el inicio hasta el final la

© Alfaomega - RC Libros

palabra clase. Siempre que se trabaje en Java se va a tener que declarar al menos una clase o una variante de una clase, pero todo parte del concepto clase.

En el código del ejemplo "Hola mundo" que estamos utilizando lo podemos ver, por más sencillo que sea la aplicación tenemos que sí o sí tener una clase y esa clase es la activity principal.

Una clase es una plantilla, por ejemplo, si yo tuviera que programar un videojuego de aviones, tendría que crear todas las clases de todo lo que va a estar en el juego, vamos a crear una clase que sea nave principal, otra que sea enemigo, otra asteroide. Dentro de cada clase se definen las características de lo que definimos, por ejemplo, una nave puede tener:

- Vida
- Tipo de armas
- Colores
- Etc.

Las clases están formadas por características, estas características en el mundo de la programación tienen un nombre y ese nombre es el atributo. Todas las características que pueda llegar a tener la clase se las conocerá formalmente como atributos y todo en Java existe gracias a la definición previa de una clase.

Las clases además de tener atributos tienen otro elemento, que son los métodos. Los métodos son las acciones. Por ejemplo, una clase nave espacial tiene estos métodos:

- Disparar
- Mover derecha
- Mover izquierda
- Despegar
- Aterrizar
- Etc.

Como podemos observar, las acciones normalmente son verbos en infinitivo y podríamos resumir el mover derecha, mover izquierda, mover arriba y mover abajo

en uno solo que puede ser mover, pero eso ya depende de la lógica de cada programador.

Nos quedamos con lo siguiente:

- En Java todo parte de una clase.
- Las clases son las definiciones o plantillas de "algo".
- Las clases tienen atributos o métodos.

Con esto podemos decir que en el código del "Hola mundo" lo que sucede es que estamos definiendo una clase que se llama "MainActivity" y que además definimos su método, onCreate. Cuando encontramos algo como @Override es una directiva que dice que el método onCreate lo estamos sobrescribiendo, es decir, que estamos escribiendo de nuevo su comportamiento.

```
public class MainActivity extends AppCompatActivity {

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

Objeto

El objeto es la manera en que te debes de imaginar el concepto sobre el cual gira la programación orientada a objetos o también conocida por su nombre corto POO. POO es una forma de programar, podemos decir que existen diferentes formas de programar y POO es la más utilizada hoy en el mundo y desde hace ya mucho tiempo. Aunque para empezar a programar se utiliza la programación estructurada y después el desarrollador da el salto a POO.

El objeto nace cuando le damos vida a una clase, todo en el mundo es un objeto, ese es el principio con el que trabaja la programación orientada a objetos, así que todo lo podemos programar, lo que necesitamos hacer primero es definir una

© Alfaomega - RC Libros

clase como ya vimos con sus atributos y con sus métodos, ya que tenemos los atributos y los métodos entonces ya tenemos todo lo necesario.

Lo que viene a continuación es crear un objeto utilizando como referencia una clase, así que decimos vamos a crear un objeto que sea un avión, como el ejemplo que hemos utilizado, por lo tanto, ese avión va a tener los atributos, los métodos que ya colocamos y también un nombre, siempre que creamos un objeto le tenemos que colocar un nombre y este debe de ser un nombre único ya que así es como nos vamos a referir a él durante todo el programa.

Pasando esto a Android podemos decir que vamos a crear objetos en relación con las miles de clases que tenemos ya definidas para Java y para Android y gracias a eso tenemos una aplicación.

Un montón de objetos colaborando entre sí hacen que tengamos desarrollada y funcionando una aplicación.

Herencia

En el código del "Hola mundo" tenemos el siguiente código:

```
public class MainActivity extends AppCompatActivity {
```

Lo que está sucediendo es que creamos una clase pública que se llama MainActivity y estamos heredando de la clase AppCompatActivity, en otras palabras, MainActivity va a ser idéntico a AppCompatActivity y que por lo tanto tendrá los mismos atributos y métodos que esta clase ya tiene definidos en su interior.

MainActivity se dice que es una clase hija de la clase AppCompatAcitivity. Existen diferentes tipos de herencia en la programación, la herencia simple y la herencia múltiple. Java no soporta la herencia múltiple, por lo tanto, la herencia que tenemos en Android es solamente la herencia simple, así que solamente se puede tener un padre, aunque muchas clases pueden seleccionar a otra clase como padre, en otras palabras, una clase puede tener muchos hijos, pero una clase hijo solo tiene un padre.

© Alfaomega - RC Libros

Haciendo un resumen del código Java del MainActivity, podemos decir que tenemos una clase MainActivity, esta hereda de la clase AppCompatactivity y que es herencia simple, solo tiene un padre. Sobrescribimos el método onCreate y por ahora no accedemos ni manipulamos ningún atributo.

Si se logró comprender esta parte de código, el lector debe estar tranquilo ya que podrá llevar los siguientes apartados del libro sin problema, en caso contrario le recomiendo volver a leer este capítulo desde el principio.

Modificadores de acceso

Una pieza importante dentro del mundo de Java en el momento de programar son los modificadores de acceso, en el ejemplo anterior mencionamos que se creaba una "clase pública" esto es en relación con la palabra public con la que se inicia la definición de la clase. En este apartado conoceremos los modificadores de acceso.

Tenemos 4 modificadores:

- Private
- Protected
- Public
- Default

Ahora explicaremos de qué trata cada uno de ellos.

Private: permite acceder solo desde la misma clase.

Protected: permite acceder desde la clase, los hijos de la clase y del mismo paquete.

Public: permite acceder desde cualquier lugar.

Default: (cuando no colocamos un modificador se toma como Default): permite acceder desde el paquete y desde la misma clase.

Estos modificadores permiten restringir el acceso a las clases, variables y métodos para poder tener claras las reglas que se necesitan seguir o respetar, en muchos casos nos sirven también como una capa extra de seguridad ya que si colocamos el modificador como debe de ser, vamos a estar protegiendo no solamente el software sino también la integridad de los datos, algo muy importante en estos días.

© Alfaomega - RC Libros

Acceder a elementos desde Java

Ya tenemos una Activity con su respectivo "Hola mundo", ahora vamos a colocar un botón y un editText. La idea principal es que coloquemos texto en el editText y que al hacer clic en el botón cambiemos el texto del TextView que dice "Hola mundo!" por el texto que está en el EditText.

Este es el código del archivo activity_main.xml:

```
<?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:id="@+id/textView"
      android:text="Hola mundo!"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
   </TextView>
   <EditText
      android:id="@+id/editText"
      android:hint="Escribe lo que sea"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
   <Button
      android:id="@+id/boton"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Cambiar texto"/>
</LinearLayout>
```

Analizaremos el código por bloques, el primer bloque es un linearlayout, lo único relevante es que tiene la orientación vertical para que los elementos que le vamos a colocar se acomoden uno debajo del anterior:

```
    <?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">
```

Ahora vemos la declaración de un elemento TextView en el que colocamos un id para poder acceder a él desde el código Java y tiene el texto "Hola mundo!" que es el que se planea cambiar por lo del EditText:

```
      <TextView
     android:id="@+id/textView"
     android:text="Hola mundo!"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
  </TextView>
```

A continuación, podemos ver el bloque del EditText en el que también ya se definió el ID y en relación con ese vamos a poder acceder al elemento desde el código Java, agregamos la propiedad hint para indicarle al usuario qué es lo que va a colocar en este cuadro:

```
      <EditText
     android:id="@+id/editText"
     android:hint="Escribe lo que sea"
     android:layout_width="match_parent"
     android:layout_height="wrap_content" />
```


© Alfaomega - RC Libros

Por último vemos el bloque del botón que es en donde tenemos un id como cualquier elemento que quiere poder ser llamado desde el código de Java, se agregó un texto que dice "Cambiar texto" con eso explicamos de forma breve qué es lo que esperamos que haga el botón al pulsarlo.

```
<Button
android:id="@+id/boton"
android:layout_width="match_parent"
android:layout_height="wrap_content"

android:text="Cambiar texto"/>
```

Esa es toda la explicación del archivo XML.

Ahora vamos a colocar el código del "Hola mundo" desde cero y le iremos agregando lo necesario para que funcione, partimos de lo que ya teníamos:

```
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);
    }
}
```

Ahora vamos a declarar los 3 elementos que en un principio son a los que necesitamos acceder.

```
TextView textView;
EditText editText;
Button button;
```

Debemos de fijarnos en que los imports aparezcan en la parte de arriba de la clase, ya que si no aparecen, nos marcará un error. Tendríamos que ver esto:

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

Nuestra clase MainActivity estaría así por ahora:

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

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

public class MainActivity extends AppCompatActivity {
    TextView textView;
    EditText editText;
    Button button;

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

    }
}
```

Ahora vamos a hacer algo muy importante, siempre que queremos utilizar desde el archivo de Java un elemento de un archivo XML tenemos que enlazar el objeto que estamos declarando aquí con el objeto colocado en el archivo XML, esto se logra con la siguiente instrucción:

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


© Alfaomega - RC Libros

Como podemos ver estamos asignándole a la variable textView el elemento textView, es muy importante que veamos que corresponde con el id que colocamos en el XML:

```
    <TextView
    android:id="@+id/textView"
    android:text="Hola mundo!"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
</TextView>
```

Como vemos el id tiene que corresponder con lo que se encuentra entre paréntesis en la instrucción findViewById().

Eso mismo tendríamos que hacer para todos los elementos que están en un archivo XML y queremos poder manipular y acceder desde el código Java:

```
editText = (EditText) findViewById(R.id.editText);
button = (Button) findViewById(R.id.boton);
```

El código del MainActivity por ahora se encuentra así:

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

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

public class MainActivity extends AppCompatActivity {
    TextView textView;
    EditText editText;
    Button button;

    @Override
```

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

        textView = (TextView) findViewById(R.id.textView);
        editText = (EditText) findViewById(R.id.editText);
        button = (Button) findViewById(R.id.boton);

    }
}
```

Ahora vamos a hacer que el botón ejecute una instrucción cuando el usuario haga clic, para eso utilizamos el método onClick, pero antes tenemos que definir la interfaz setOnClickListener. El código es este:

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String texto = editText.getText().toString();
        textView.setText(texto);
    }
});
```

Al botón que ya teníamos definido le implementamos el setOnClickListener y dentro definimos el OnClick.

Lo que hacemos dentro del OnClick es crear una cadena que llamamos texto a la que le colocamos lo que tiene el editText, eso lo logramos gracias al método getText () pero antes debemos de asegurarnos de que es una cadena de texto lo que estamos colocando dentro, por eso utilizamos la conversión toString ().

Finalmente, al texView que ya tenemos declarado y listo para utilizar desde el código Java le decimos que cambie el texto que tiene por lo que colocamos en la variable "texto" que es la cadena que se llenó con el contenido del editText.

Así queda nuestro código finalmente:

© Alfaomega - RC Libros

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

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

public class MainActivity extends AppCompatActivity {
    TextView textView;
    EditText editText;
    Button button;

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

        textView = (TextView) findViewById(R.id.textView);
        editText = (EditText) findViewById(R.id.editText);
        button = (Button) findViewById(R.id.boton);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String texto = editText.getText().toString();
                textView.setText(texto);
            }
        });
    }
}
```

Ahora vamos a probar nuestra aplicación y ver el resultado, así se ve nuestra aplicación antes de ejecutar:



Ahora escribimos texto dentro del EditText:



© Alfaomega - RC Libros

Ahora hacemos clic en el botón que dice cambiar texto y veremos el resultado:



El texto se cambió por lo que habíamos colocado en el EditText, hemos logrado la primera interacción entre elementos, tenemos un botón, un editText y un TextView.

Toast

El elemento Toast nos sirve para colocar un mensaje dentro de una aplicación hacia el usuario, este mensaje tiene 2 medidas de tiempo para su duración: corta o larga.

Vamos a agregar un botón nuevo a lo que ya teníamos del archivo XML del ejemplo anterior y que contenga el texto "Mostrar mensaje". Así quedaría el archivo XML:

```
<?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:id="@+id/textView"
        android:text="Hola mundo!"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </TextView>
    <EditText
        android:id="@+id/editText"
        android:hint="Escribe lo que sea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/boton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Cambiar texto"/>
    <Button
        android:id="@+id/msj"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Mostrar Mensaje"
        />
</LinearLayout>
```

Ahora vemos que al final del archivo tenemos un botón más que tiene el id "msg" y el texto "Mostrar mensaje". Ahora lo que tenemos que hacer es definir el nuevo botón que agregamos:

```
Button button,msg;
```

A continuación, referenciamos el elemento que definimos en el código Java con el que se encuentra en el archivo XML:

© Alfaomega - RC Libros

```
textView = (TextView) findViewById(R.id.textView);
editText = (EditText) findViewById(R.id.editText);
button = (Button) findViewById(R.id.boton);
msg =  (Button) findViewById(R.id.msg);
```

Agregaremos el onClick al botón que tiene el nombre "msg":

```
msg.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


    }
});
```

Dentro del onClick vamos a colocar cómo utilizaremos el Toast:

```
    Toast.makeText(getApplicationContext(),"Esto es un
mensaje",Toast.LENGTH_LONG).show();
```

Veamos en qué consiste el código, primero utilizamos el elemento Toast con el método makeText, a este método le tenemos que pasar tres parámetros:

- El contexto
- El texto que va a mostrar
- El tiempo que va a durar el mensaje

Todos los parámetros van separados con comas, el contexto lo obtenemos de nuestra aplicación con la instrucción getApplicationContext (), después colocamos entre comillas el mensaje, en este caso el mensaje es "Esto es un mensaje" pero el lector sin ningún problema puede cambiar este mensaje por el que crea conveniente.

Finalmente colocamos el tiempo de duración y dentro de Toast ya contamos con la duración, larga o corta. Si es larga la que necesitamos, colocamos Toast.LENGTH_LONG, si es corta la que necesitamos, colocamos Toast.LENGTH_SHORT, después de colocar la duración y de cerrar el paréntesis es importante colocar una última instrucción que es show(), sin esta instrucción

simplemente no podremos ver nada y aunque todo esté bien, el mensaje no se mostrará en nuestra aplicación. Así queda finalmente nuestra MainActivity:

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

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

import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    TextView textView;
    EditText editText;
    Button button,msg;



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

        textView = (TextView) findViewById(R.id.textView);
        editText = (EditText) findViewById(R.id.editText);
        button = (Button) findViewById(R.id.boton);
        msg =  (Button) findViewById(R.id.msg);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String texto = editText.getText().toString();
                textView.setText(texto);
            }
        });
```


© Alfaomega - RC Libros

```
        msg.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(),"Esto es un
mensaje",Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Observemos cómo se ve nuestra aplicación después de hacer clic:



Ahora cuando hacemos clic en el botón que acabamos de agregar vamos a ver el mensaje de la siguiente forma:



En la parte inferior podemos ver el mensaje, y la duración la podemos modificar con los valores que ya hemos comentado anteriormente.

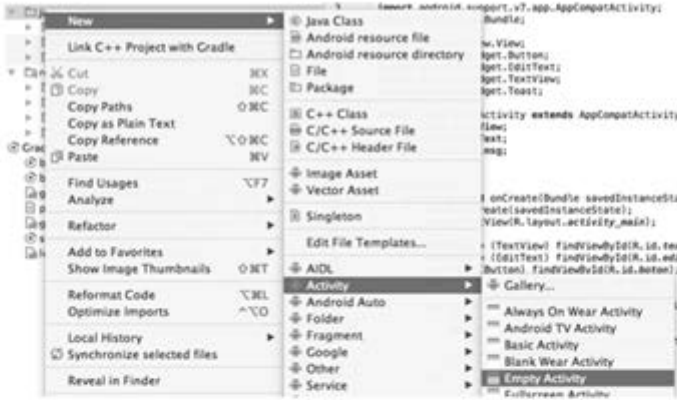
© Alfaomega - RC Libros

15 AGREGAR UNA ACTIVITY

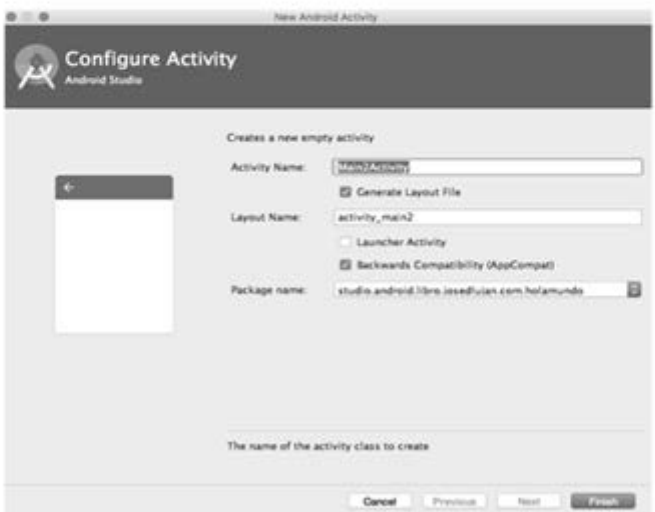
En este capítulo vamos a agregar una activity y después haremos que nuestro MainActivity que ya tenemos definido nos mande a la nueva activity que hemos creado, para eso es muy importante definir en qué consiste crear una activity y qué es lo que tenemos que hacer.

Para agregar una activity tenemos diferentes opciones, pero tomaremos la opción más sencilla por su velocidad y también por su facilidad para recordar, por lo tanto, vamos a colocarnos con el ratón sobre el directorio "java" y vamos a dar clic en el botón derecho, seleccionaremos New y después Activity y finalmente Empty Activity:

Java -> New -> Activity -> Empty Activity



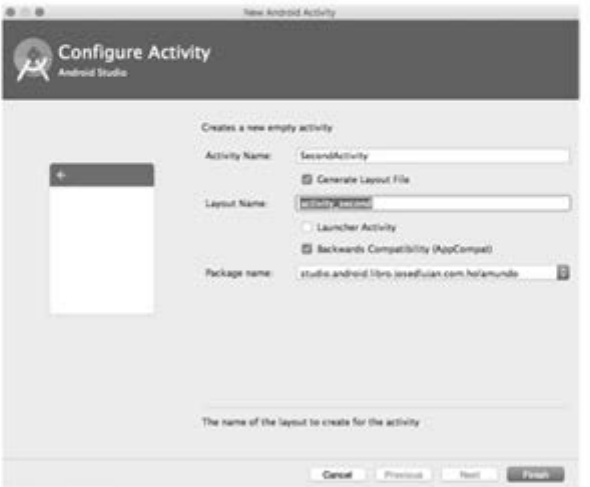
Ahora veremos la siguiente ventana:



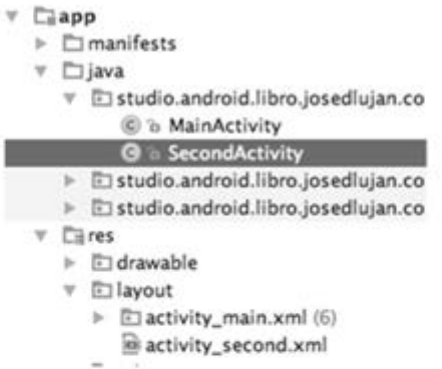
© Alfaomega - RC Libros

Nos pedirá los siguientes datos: Activity Name. Debemos de colocar el nombre de la nueva actividad y debemos de seguir las reglas ya conocidas para nombrar la actividad, en mi caso le colocaré SecondActivity. A continuación, nos dice en un cuadro de selección si queremos crear el archivo xml, lo dejamos seleccionado para que lo haga.

En el otro cuadro nos pide el nombre del layout, por convención se maneja un nombre que se parezca al primero, así que yo colocaré el nombre activity_second, también dejamos seleccionada la siguiente casilla que dice si utilizaremos AppCompat, finalmente nos pide el nombre del paquete al que va a pertenecer, deberíamos de dejar en la mayoría de los casos el que tiene por defecto. Hacemos clic en el botón "Finish" y listo.



Ahora vemos los archivos en el lado izquierdo de nuestro proyecto:



Veremos el contenido del archivo SecondActivity.java:

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

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

public class SecondActivity extends AppCompatActivity {

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

Veamos el archivo XML, activity_second.xml

```
<?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_second"
```


© Alfaomega - RC Libros

```
    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.SecondActivity">

</RelativeLayout>
```

Por defecto nos coloca un RelativeLayout, pero por ahora eso no es relevante.

Invocar una Activity

Veamos cómo una activity puede mandar a llamar a otra activity. En este caso tomaremos la segunda activity como la activity que va a ser llamada. Para llamar a otra activity agregaremos un botón más al XML que ya tenemos creado y le ordenaremos que al hacer clic invoque la nueva activity.

Este seria el código del archivo XML con el nuevo botón:

```
<?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:id="@+id/textView"
        android:text="Hola mundo!"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </TextView>
    <EditText
```

```
        android:id="@+id/editText"
        android:hint="Escribe lo que sea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/boton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Cambiar texto"/>
    <Button
        android:id="@+id/msg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Mostrar Mensaje"
        />
    <Button
        android:id="@+id/nuevaActivity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Nueva Activity"/>
</LinearLayout>
```

Al final del archivo hemos agregado el botón que tiene como id "nuevaActivity" y además tiene el texto "Nueva Activity", esos serían los únicos cambios en relación con su última versión.

Ahora vamos a ir al archivo MainActivity.java, este es el código:

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

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

import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
```


© Alfaomega - RC Libros

```
public class MainActivity extends AppCompatActivity {
    TextView textView;
    EditText editText;
    Button button,msg,nuevaActivity;



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

        textView = (TextView) findViewById(R.id.textView);
        editText = (EditText) findViewById(R.id.editText);
        button = (Button) findViewById(R.id.boton);
        msg =  (Button) findViewById(R.id.msg);
        nuevaActivity = (Button) findViewById(R.id.nuevaActivity);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String texto = editText.getText().toString();
                textView.setText(texto);
            }
        });

        msg.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(),"Esto es un
mensaje",Toast.LENGTH_LONG).show();
            }
        });

        nuevaActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
            Intent = new Intent(MainActivity.this,SecondActivity.class);
            startActivity(intent);
        }
    });
  }
}
```

Podemos ver en el código que tenemos agregado el nuevo elemento botón en la definición:

```
    Button button,msg,nuevaActivity;
```

Lo relacionamos con el elemento del archivo XML:

```
        nuevaActivity = (Button) findViewById(R.id.nuevaActivity);
```

Definimos un onClickListener y un onClick para el botón:

```
        nuevaActivity.setOnClickListener(new View.OnClickListener()
{
        @Override
        public void onClick(View v) {


        }
    });
```

Intent

Dentro del método onClick tenemos un nuevo elemento del que no hemos hablado que es Intent.

© Alfaomega - RC Libros

El intent es un elemento muy sencillo de utilizar, pero el concepto y otras cosas que lo rodean lo convierten en un principio muy abstracto y por lo tanto difícil de comprender, más cuando nos estamos iniciando en el mundo del desarrollo Android, en este apartado se tratará de dar la información de la forma más clara, pero también comentaremos que el intent tiene distintos usos y caras que poco a poco irá el desarrollador relacionando conforme avance en su formación.

Un intent lo podemos definir como un "intento" de hacer algo o como una operación, al ser una operación o un "intento" significa que podemos hacer diferentes cosas. El intent siempre va acompañado de información que nos indica qué es lo que esperamos que el sistema operativo Android haga.

En este caso vamos a utilizar el intent para que haga una llamada a la otra Activity y que esta se muestre al usuario, como cualquier objeto en Java vamos primero a crear el objeto y vamos a pasarle en este caso 2 parámetros que necesitamos, uno es el contexto y el otro la clase que queremos invocar:

```
Intent intent = new Intent(MainActivity.this,SecondActivity.class);
```

El contexto lo estamos obteniendo de la misma activity en donde se encuentra la instrucción de crear el intent que es MainActivity, para acceder a su contexto le indicamos con el nombre de la Activity y seguida de la palabra this.

Por último ejecutamos la instrucción startActivity a la que le tenemos que pasar un intent, este intent es el que previamente creamos con los 2 parámetros ya mencionados.

```
startActivity(intent);
```

Así veremos la Activity principal con el cambio del nuevo botón:



Cuando pulsamos en el botón "Nueva Activity" deberíamos de ver la siguiente activity:



© Alfaomega - RC Libros

La actividad está vacía pero el lector podría colocar cualquier elemento para que visualmente le sea más fácil identificar, pero como sabemos la actividad que creamos solamente contiene un RelativeLayout, que en este caso podemos ver que no tiene nada.

Intent para llamadas

Vamos a agregar un botón que tenga la capacidad de hacer una llamada, este será el encargado de realizar la llamada en el momento de ser presionado el número que coloquemos en el código, este ejercicio se puede complicar un poco si hacemos uso de un *EditText* y tomar de ahí el número.

El código de la interfaz gráfica es el siguiente, utilizaremos el archivo activity_second.xml:

```
    <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"
        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.SecondA
ctivity" >

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

    </RelativeLayout>
```

El *RelativeLayout* cuenta con un ancho y alto del tamaño del padre, dentro de los atributos que colocamos tenemos el *padding* para separar los elementos que

estarán contenidos. Agregamos un botón al que le ponemos un *ID "boton"* y el texto lo tomamos de la cadena de texto denominada *"llamar"*.

El código del *MainActivity.java* es el siguiente:

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

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

public class SecondActivity extends AppCompatActivity {
    Button boton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        boton = (Button) findViewByld(R.id.boton);
        boton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                llamar();
            }
        });
    }

    private void llamar(){
        Uri uri = Uri.parse("tel:9981663748");
        Intent intent = new Intent(Intent.ACTION_DIAL,uri);
        startActivity(intent);
    }

}
```

Analizaremos el código por partes, tenemos declarada la actividad haciendo herencia del AppCompatActivity:

© Alfaomega - RC Libros

```
public class SecondActivity extends AppCompatActivity {
```

Enlazamos el elemento botón que declaramos con el que tenemos en el XML.

```
boton = (Button) findViewByld(R.id.boton);
```

Definimos el onClickListener y también el onClick:

```
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


    }
```

Algo que no habíamos hecho hasta este apartado del libro es llamar a un método, todo lo habíamos hecho colocando el código que queremos ejecutar dentro del onClick, ahora vamos a cambiar esto solamente llamando a un método y que este método contenga el código que ejecutará la llamada:

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

Si vemos el método, lo que contiene primero es el uso de la clase Uri y una de las cosas que nos permite es colocarle el prefijo "tel" seguido de un número al que nos interese en este caso hacer una llamada:

```
Uri uri = Uri.parse("tel:9981663748");
```

Después vemos en el método que tenemos la definición de un intent y este a diferencia del que usamos anteriormente tiene una instrucción que es Intent.ACTION_DIAL, esta instrucción nos permite invocar una instrucción ya predefinida para hacer una llamada:

```
Intent intent = new Intent(Intent.ACTION_DIAL,uri);
```

Finalmente ejecutamos el intent al igual que en el caso anterior en donde lo usamos, con la instrucción startActivity:

```
startActivity(intent);
```

Permisos para llamar

Cuando queremos acceder a alguno de los elementos del hardware del teléfono o también a ciertas funcionalidades o información del teléfono como los contactos, mensajes o llamar, no debemos olvidar modificar el *manifest* y agregar el permiso correspondiente, en este caso el permiso que necesitamos es el de *"llamar"*, y agregaríamos la siguiente línea al *manifest*:

```
<uses-permission
android:name="android.permission.CALL_PHONE"/>
```

Esta línea se recomienda colocarla antes de la etiqueta *"application"*. El archivo AndroidManifest.xml quedaría de la siguiente forma:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

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

  <uses-permission android:name="android.permission.CALL_PHONE"
/>
   <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"
```


© Alfaomega - RC Libros

```
            android:configChanges="orientation|keyboardHidden"
            android:screenOrientation="portrait" />

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

</manifest>
```

Abrimos la segunda Activitiy ya que ahí es donde agregamos el botón y se debe de ver de la siguiente forma:



Al pulsar el botón que dice "Llamar" vamos a ver la siguiente ventana:



Podemos ver que es una pantalla de marcado en la que se instaló de manera automática el número que colocamos, ahora ya solo tendríamos que marcar, pero esto no es una llamada automática, si queremos que el número sea marcado sin que nosotros tengamos que hacer clic al botón de llamar, podemos cambiar la instrucción del intent por la siguiente:

```
Intent intent = new Intent(Intent.ACTION_CALL,uri);
```

Además de eso tendríamos que hacer la verificación de los permisos y ver si se encuentran los permisos necesarios:

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

Intent para enviar SMS

Vamos a cambiar el uso del botón que ya tenemos para llamar por uno que envíe mensajes. La interfaz gráfica está compuesta por un *RelativeLayout* que

© Alfaomega - RC Libros

contiene un botón, este incluye el texto *"Enviar SMS"* y al recibir el clic enviará un sms, también conocido como mensaje de texto, el lector puede intentar mejorar el reto al utilizar *EditText* como en el apartado anterior para colocarlo y además manipularlo como cualquier cadena de texto.

Este es el código del archivo XML:

```
    <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"
        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.SecondA
ctivity " >

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

    </RelativeLayout>
```

El código del archivo *SecondActivity.java* es el siguiente:

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

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

```
public class SecondActivity extends AppCompatActivity {
    Button boton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        boton = (Button) findViewById(R.id.boton);
        boton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                enviar();
            }
        });
    }

    private void enviar(){

        Uri uri = Uri.parse("smsto:9981642758");
        Intent intent = new Intent(Intent.ACTION_SENDTO,uri);
        intent.putExtra("sms_body", "Hola mundo de msg");
        startActivity(intent);



    }

}
```

Definimos el botón que vamos a utilizar en el código:

```
Button boton;
```

Ahora enlazamos el elemento botón del código del XML:

```
boton = (Button) findViewById(R.id.boton);
```


© Alfaomega - RC Libros

Definimos el setOnClickListener y el Onclick con el método enviar:

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

El método enviar tiene en su primera línea el método Uri, que de igual manera que al llamar a otro teléfono nos permite colocar la cadena "smsto" y podremos enviar el mensaje al número después de esta cadena de texto:

```
        Uri uri = Uri.parse("smsto:9981642758");
```

En el intent colocamos la instrucción ACTION_SENDTO que es la que indica a Android que el intent tiene que enviar el mensaje.

```
        Intent intent = new Intent(Intent.ACTION_SENDTO,uri);
```

Ahora vamos a colocar un extra, que es el texto que vamos a enviar en el mensaje:

```
        intent.putExtra("sms_body", "Hola mundo de msg");
```

Ahora invocamos la ejecución del intent con la instrucción startActivity:

```
        startActivity(intent);
```

© Alfaomega - RC Libros

Así es como se verá nuestra activity que tiene el botón enviar:



En el momento de pulsar el botón enviar aparece la siguiente ventana:



Vemos que el mensaje ya está listo para enviar, tiene el número en el apartado correspondiente y tiene el texto ya escrito en donde deberíamos de escribir el mensaje.

© Alfaomega - RC Libros

16 WEBVIEW

En este capitulo vamos a ver un elemento que es muy sencillo y de los más utilizados en Android que es el WebView. Este como todos los elementos es una clase que nos proporciona Android para usarla como base y crear nuestro propio navegador o también podemos utilizarla para ver archivos *html* en nuestro navegador. Esta clase utiliza el motor de renderizado Webkit, incluye todas las características básicas de un navegador como zoom, búsquedas de texto, desplazamiento atrás y adelante, entre otros.

El XML que vamos a utilizar para el *WebView* debe tener el siguiente 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="studio.android.libro.josedlujan.com.holamundo.SecondA
ctivity">

        <WebView
           android:id="@+id/webview"
           android:layout_width="match_parent"
           android:layout_height="match_parent"
           />

    </LinearLayout>
```

Cambiamos el clásico RelativeLayout por un LinearLayout solo para variar y que el lector vea que es lo mismo usar uno u otro. Al elemento *WebView* de nuestro XML lo expandimos al tamaño total de nuestra pantalla y para eso los atributos *layout_width* y *layout_height* los colocamos con el valor *match_parent*.

Finalmente le colocamos un id que es webview para poder acceder a este elemento desde el archivo Java.

La activity tiene el siguiente código:

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


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.WebResourceRequest;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class SecondActivity extends AppCompatActivity {


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
      webView = (WebView) findViewById(R.id.webview);
      webView.getSettings().setJavaScriptEnabled(true);
      webView.loadUrl("http://josedlujan.com");
      webView.setWebViewClient( new  WebViewClient()
      {

         public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request)
         {
            view.loadUrl(request.getUrl().toString());
```


© Alfaomega - RC Libros

```
            return false;

        }


    });

  }
}
```

Declaramos un elemento *WebView*:

```
    WebView webView;
```

Enlazamos el elemento *webView* que ya declaramos con el elemento *WebView* del elemento XML:

```
    webView = (WebView) findViewById(R.id.webview);
```

Utilizamos una de las propiedades más demandadas de un navegador que es la utilización de javascript, en la siguiente línea habilitamos el uso de *JavaScript* en nuestro elemento, en caso de no querer permitir su uso cambiaríamos el valor que encontramos ahora en "true" por "false":

```
    webView.getSettings().setJavaScriptEnabled(true);
```

Colocamos la url de la web que queremos que cargue el webview y utilizamos la instrucción loadUrl:

```
    webView.loadUrl("http://josedlujan.com");
```

Con lo que hemos descrito por separado ya debe de funcionar nuestra aplicación, pero el siguiente bloque de código es el que le agrega una funcionalidad

© Alfaomega - RC Libros

importante a nuestro webview, cuando pulsamos un enlace, este se va a cargar en otro navegador, es decir, que lo abriríamos en el navegador de nuestro dispositivo Android y por lo tanto podemos decir que abrimos dos navegadores, este funcionamiento no sería adecuado, por ello colocamos el siguiente código:

```
webView.setWebViewClient( new  WebViewClient()
{

   public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request)
   {
      view.loadUrl(request.getUrl().toString());
      return false;

   }



});
```

El código anterior hace la carga del enlace al que se le dio sobre el mismo webview, así que evitamos abrir un nuevo navegador, esta sería la forma correcta de manejar la interacción entre el usuario y el sitio web que estamos colocando en el webview.

Algo que debemos siempre recordar es que necesitamos habilitar el permiso que precisamos para poder usar internet, como cualquier permiso en Android este se encuentra en el Manifest, tendremos que agregar esta línea:

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

Así queda el archivo Manifest:

© Alfaomega - RC Libros

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

    <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"
                    android:configChanges="orientation|keyboardHidden"
                    android:screenOrientation="portrait" />

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

</manifest>
```

17 AUDIO

En Android tenemos distintas formas de reproducir sonidos dentro de una aplicación.

Tenemos 2 API's principales que nos permiten manejar los sonidos: Soundpool y MediaPlayer.

1. *Soundpool* se utiliza para sonidos cortos y menos de 1 MB, además puede ejecutar varios a la vez.
2. *MediaPlayer* es para sonidos largos y también podemos usarla para videos, películas, etc.

En este capítulo veremos la más usada que es MediaPlayer.

MediaPlayer

Para el ejercicio del MediaPlayer vamos a crear una interfaz con 2 botones, uno para iniciar la reproducción del audio y otro para detenerlo.

El código del archivo XML es:

```
<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.SecondA
ctivity" >

    <Button
        android:id="@+id/reproducir"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Reproducir"
        />
    <Button
        android:id="@+id/parar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Parar"
        />


</LinearLayout>
```

Directorio raw

El directorio raw es un directorio que se coloca dentro del directorio res, su principal objetivo es colocar archivos que tienen formatos para video, audio, y también puede que otro formato de texto.

Existe un directorio muy similar que es el directorio assets, este directorio también permite colocar recursos, aunque es más popular para colocar, por ejemplo, bases de datos o algunas imágenes.

La diferencia entre el directorio assets y el directorio raw, es que en este último cuando colocamos un elemento sin importar lo que sea le estamos diciendo a Android Studio que este archivo no se va a compilar, no es partícipe de este proceso; por lo tanto, se coloca dentro del paquete de forma directa, se incluye sí o sí. Además, se le asigna un identificador, este identificador lo podemos ver como cualquier otro elemento al que ya se le asignó en el famoso archivo R.java, que es donde se tienen todos los identificadores.

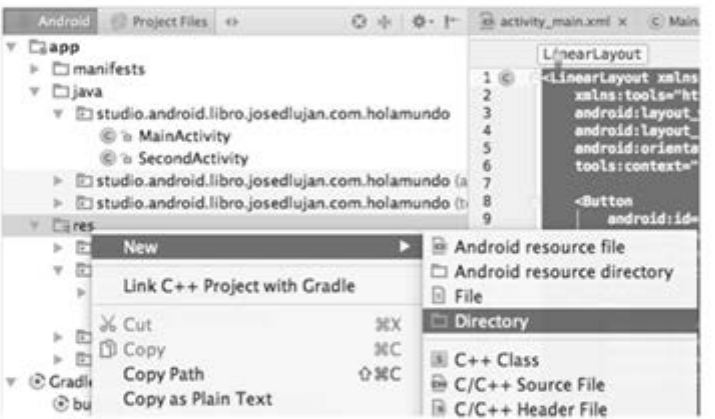
© Alfaomega - RC Libros

Para colocar el archivo de música vamos a buscar cualquier archivo .MP3 y lo colocamos en un directorio raw que vamos a crear.

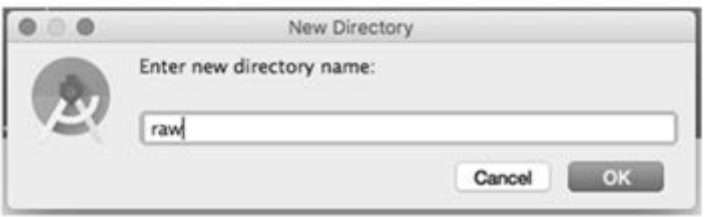
Vamos a pulsar en el directorio res, hacemos clic con el botón derecho y pulsamos NEW, luego a directory y le damos el nombre de raw.

```
res -> new -> directory
```

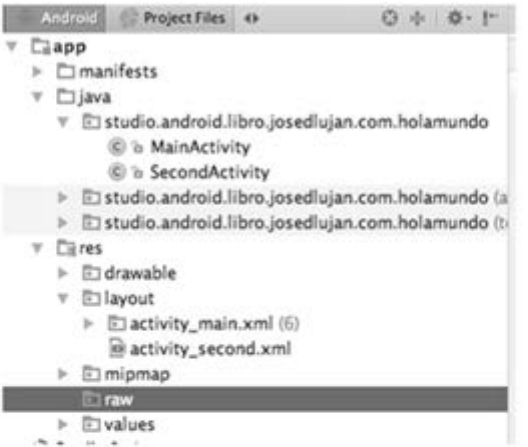
Se ve así:



Colocamos el nombre y listo:



Tenemos entonces ya el directorio raw:



Colocamos el archivo mp3 que queremos reproducir y lo arrastramos al directorio raw, y al soltarlo deberíamos de ver el archivo dentro del directorio:



© Alfaomega - RC Libros

Ahora vemos el código del archivo que reproducirá la música, es el archivo SecondActivity.java:

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



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

public class SecondActivity extends AppCompatActivity {
    Button reproducir,parar;
    public MediaPlayer mp;


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

        reproducir = (Button) findViewById(R.id.reproducir);
        parar = (Button) findViewById(R.id.parar);

        reproducir.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mp =
MediaPlayer.create(getApplicationContext(),R.raw.musica);
                mp.start();


            }
        });

        parar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
            mp.stop();
        }
    });
  }

}
```

Veamos el código por partes, en la primera parte solamente tenemos los imports necesarios para nuestra aplicación:

```
import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

Definimos 3 elementos que vamos a usar, el elemento MediaPlayer y otros 2 elementos que son los botones, uno para iniciar el audio y otro para detenerlo.

```
Button reproducir,parar;
public MediaPlayer mp;
```

Enlazamos los botones con sus respectivos elementos en el archivo XML:

```
reproducir = (Button) findViewById(R.id.reproducir);
parar = (Button) findViewById(R.id.parar);
```

Definimos primero el onClickListener y el onClick del botón Reproducir, aunque no es de vital importancia el orden, ya que son eventos y se mandarán llamar únicamente cuando sean necesarios.

```
        reproducir.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


    }
});
```


© Alfaomega - RC Libros

Dentro del elemento onClick definimos la creación del MediaPlayer, le tenemos que pasar 2 parámetros, el contexto y el archivo de audio que vamos a reproducir, el contexto lo obtenemos utilizando getApplicationContext, el archivo de audio está en la ruta R.raw.musica, estamos haciendo referencia al archivo R como cualquier elemento:

```
    MediaPlayer.create(getApplicationContext(),R.raw.musica);
            mp.start();
```

Definimos el onClickListener y el onClick del botón parar:

```
parar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {


        }
    });
  }

}
```

Dentro del onClick solo tenemos que llamar a la instrucción detener.

```
                mp.stop();
```

Con esto terminamos la reproducción del audio en caso de que esté la aplicación reproduciendo el audio.

© Alfaomega - RC Libros

18 SENSORES

En este capitulo de sensores aprovecharemos las últimas actualizaciones que ha tenido Android Studio y que han agregado algo muy interesante al emulador, desde la versión 2.2 de Android Studio podemos ahora probar algunos sensores que antes solo podiamos probar en un teléfono fisico.

El sensor con el que vamos a trabajar es el acelerómetro. Este es un sensor de los muchos que actualmente vienen en la mayoria de los dispositivos Android, permite medir la aceleración dependiendo del movimiento, tiene una posición que es la de una ventana como la referencia para tomar la medida de ahí.

En nuestro ejemplo tendremos una pantalla de Android con 3 TextView que van a cambiar los valores dependiendo del movimiento que se haga y que el acelerómetro registre. Así que los TextViews se estarán alimentando de los valores que devuelva el acelerómetro.

Vamos a ver el código del archivo XML:

```
<?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="josedlujan.acelerometro.ejemplo.ejemploacelerometro.MainActivity">

    <TextView
        android:id="@+id/x"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <TextView
        android:id="@+id/y"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <TextView
        android:id="@+id/z"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

En el código anterior tenemos un LinearLayout que es el contenedor padre de todos los elementos y dentro tiene 3 TextView, uno con el nombre de "x", otro "y" y el último con el nombre de "z", haciendo referencia cada uno de los nombres al eje del que va a mostrar el valor. Esto quiere decir que tenemos 3 ejes, X,Y,Z.

Ahora veamos el código del archivo Java de nuestra aplicación:

```
package josedlujan.acelerometro.ejemplo.ejemploacelerometro;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```


© Alfaomega - RC Libros

```
import java.util.List;

public class MainActivity extends AppCompatActivity implements
SensorEventListener {
    TextView x,y,z;
    Sensor acelerometro;
    SensorManager sm;
    List<Sensor> sensors;

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

        x = (TextView) findViewById(R.id.x);
        y = (TextView) findViewById(R.id.y);
        z = (TextView) findViewById(R.id.z);

        sm = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
        //acelerometro

        if(sensors.size()>0)
            sm.registerListener((SensorEventListener)
this,sensors.get(0),SensorManager.SENSOR_DELAY_NORMAL);



    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        x.setText("Valor de x"+event.values[0]);
        y.setText("Valor de y"+event.values[1]);
        z.setText("Valor de z"+event.values[2]);
```

```
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
}
```

En la primera parte hacemos la herencia como cualquier activity en la realidad de la clase AppCompartActivity, pero implementamos la interfaz SensorEventListener, esta interfaz nos permitirá escuchar lo que el sensor escucha y utilizarlo para colocarlo en el TextView correspondiente.

```
public class MainActivity extends AppCompatActivity implements
SensorEventListener {
```

Ahora tendremos tres elementos textview X,Y,X definidos. Además, definimos el sensor que utilizamos y le damos el nombre de acelerómetro, la clase Sensor es una clase genérica que nos sirve para cualquier sensor, por eso le ponemos un nombre referente al tipo de sensor que es. Definimos el SensorManager que es el administrador de los sensores y finalmente definimos una lista del tipo Sensor ya que la utilizaremos más adelante para hacer algunas consultas.

```
TextView x,y,z;
Sensor acelerometro;
SensorManager sm;
List<Sensor> sensors;
```

Definimos a continuación los 3 textview haciendo referencia a los TextView dentro del archivo XML, le indicamos al SensorManager que necesitamos funcione como un SENSOR_SERVICE, así comenzamos la configuración del SensorManager.

Al sensor le decimos que lo vamos a utilizar como un sensor del tipo acelerómetro pero estamos preguntando al administrador de sensores que lo traiga de esa forma para comenzar a utilizarlo:

© Alfaomega - RC Libros

```
x = (TextView) findViewById(R.id.x);
y = (TextView) findViewById(R.id.y);
z = (TextView) findViewById(R.id.z);


sm = (SensorManager) getSystemService(SENSOR_SERVICE);
sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
```

Ahora verificamos el sensor en la propiedad de tamaño, la idea es que si tenemos un valor mayor que cero si tenemos disponible el sensor acelerómetro para usarlo, en caso contrario no lo tenemos y no podremos realizar el ejercicio.

```
if(sensors.size()>0)
```

Si la instrucción anterior da verdadero, entonces entramos en el if y ejecutamos la instrucción, preparamos el listener que nos va a permitir estar atento de los cambios y valores del sensor, le pasamos los parámetros como el contexto, le pedimos que tome la posición cero y le indicamos la velocidad con la que vamos a manejar los valores, que es la velocidad "retraso normal".

```
sm.registerListener((SensorEventListener)
this,sensors.get(0),SensorManager.SENSOR_DELAY_NORMAL);
```

En el manejo de este sensor tenemos 2 métodos, el método onSensorChanged y onAccuracyChanged. El primero es el método que se manda a llamar cada vez que el sensor detecta un valor nuevo, el segundo método se manda a llamar cada vez que el sensor maneja diferente precisión.

Como queremos que se refleje el cambio de valores dentro de los TextView utilizamos entonces el método onSensorChanged para decirle a los TextView que tomen el valor de un arreglo, el arreglo se llama event y este nos llega por el evento, únicamente tenemos que saber que la posición 0 es el valor X, la posición 1 es la de Y, finalmente la posición 2 es la de la letra Z.

```
@Override
public void onSensorChanged(SensorEvent event) {
    x.setText("Valor de x"+event.values[0]);
    y.setText("Valor de y"+event.values[1]);
    z.setText("Valor de z"+event.values[2]);


}


@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {


}
```

Así vemos en un principio nuestra interfaz:



Podemos mover el teléfono desde el emulador utilizando una opción, la flecha en la siguiente imagen señala la opción:

© Alfaomega - RC Libros



Cuando hacemos clic entonces veremos un menú en donde podemos encontrar la opción del emulador. La opción es la virtual sensors.



Cuando movemos el emulador por ejemplo así:



Podemos ver que los valores de X,Y,Z han cambiado en sus respectivos TextViews.



Cada vez que hagamos un movimiento del teléfono podremos ver cómo cambian los valores.

© Alfaomega - RC Libros

19 FRAGMENTOS

Android en un principio fue un sistema operativo diseñado para teléfonos móviles, durante un tiempo esto siguió igual, hasta que apareció el iPad y la mayoría de los fabricantes quisieron entrar en el mercado de las tablets, así que se comenzaron a lanzar tablets, pero desafortunadamente Android aún no estaba preparado para eso.

La versión que se lanzó para mejorar la experiencia de usuario de las tablets con Android, es la versión *AndroidHoneyComb* 3.0 que es la *API* 11.11.

En la versión 3.0 se agregó un concepto nuevo que es el "fragmento" o *fragment* en inglés. El fragmento lo podemos definir como una porción de la interfaz que tiene sus propios comportamientos, además de que tiene su ciclo de vida. La idea es tener 2 o más fragmentos conviviendo en la misma pantalla y que interactúen entre ellos, en el caso de que se necesite otro fragmento que no se está mostrando, deberíamos hacer desaparecer uno de los que tenemos y mandar a llamar a otro para mostrarlo al usuario. Esto permite una gran flexibilidad en el diseño y usabilidad para el usuario.

Si queremos crear fragmentos y que estos funcionen en versiones anteriores a la *API* 11, tendremos que utilizar la librería de soporte, pero como en la actualidad el uso de *API's* menores era pequeño, lo recomendable es desarrollar para la versión 4. Esto lo podemos sustentar con la documentación oficial de Android. Así que ya no tendremos que preocuparnos por la compatibilidad con versiones anteriores.

En realidad, crear un fragmento no debe causar problemas si hemos podido llegar a este capítulo y realizar los ejemplos. Pero uno de los problemas más interesantes durante la implementación de fragmentos es la comunicación entre ellos. Ese es el problema que vamos a abordar en este capítulo.

Vamos a crear una actividad con dos fragmentos y estos se van a comunicar entre ellos de forma correcta, ya que se pueden comunicar de varias formas, pero no todas son correctas.

Tendremos una interfaz principal con un *LinearLayout* común y dentro de esta utilizamos dos elementos del tipo *"fragment"*, uno que llamaremos izquierda y otro derecha, vamos a colocarle un peso total al contenedor principal para dividir después el peso entre los fragmentos y que estos ocupen el 50% de la pantalla cada uno. Al final en el atributo *"class"* de cada uno de los fragmentos tendremos que apuntar al archivo en el que se encuentra su código *.java* o a la clase que los define en código.

Tenemos nuestro archivo de la interfaz principal, el *activity_main.xml*:

```
    <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="horizontal"
        android:weightSum="1"
        tools:context="com.example.ejemplofragmentos.MainActivity" >

      <fragment
         android:id="@+id/izquierda"
         android:layout_width="0dp"
         android:layout_weight=".5"
         android:layout_height="match_parent"
         class="com.example.ejemplofragmentos.Izquierda"
         />
      <fragment
         android:id="@+id/derecha"
         android:layout_width="0dp"
         android:layout_weight=".5"
         android:layout_height="match_parent"
```


© Alfaomega - RC Libros

```
        class="com.example.ejemplofragmentos.Derecha"
        />
    </LinearLayout>
```

Creando la interfaz gráfica de los fragmentos

Para crear el fragmento primero vamos a crear la interfaz y después el código, algo muy parecido a lo que ya hicimos anteriormente con las actividades.

Creamos un archivo XML dentro del directorio *layout* que será la interfaz gráfica, para eso hacemos clic derecho en el directorio *layout* y después en *New* y finalmente *Android*.

Camino: **Clic derecho directorio layout -> New -> Android (nombre derecha.xml) (nombre derecha.xml)**

Lo llamamos *derecha.xml* y solo le colocamos un *LinearLayout* con orientación vertical, dentro le colocamos un elemento *TextView* al que le ponemos un texto *"Hola a todos"*, y el código queda de la siguiente forma:

```
    <?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:id="@+id/txt"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hola a todos!!!"
        />
    </LinearLayout>
```

Repetimos el mismo procedimiento que utilizamos para crear el archivo *derecha.xml* pero ahora lo llamaremos *izquierda.xml*, este último será el fragmento de la parte izquierda de la pantalla. Este fragmento contendrá un EditText que es en el que vamos a colocar texto y además tendrá un botón, el cual se encargará de

tomar el texto escrito en el *EditText* y enviarlo al segundo fragmento que tendría que recibirlo y colocarlo en el *TextView*, así realizaremos la comunicación entre estos dos fragmentos.

La interfaz de izquierda.xml quedaría de la siguiente forma:

```
<?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" >


    <EditText
        android:id="@+id/campotxt"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
        />

    <Button
      android:id="@+id/boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/boton"
            />

</LinearLayout>
```

Creando la interfaz de comunicación

Para comunicar los fragmentos de la manera correcta tendremos que crear una interfaz, este término no se debe confundir con el término interfaz gráfica. La interfaz en Java significa una colección de métodos abstractos y propiedades, esto quiere decir que se especifican pero no se implementan, el término de implementación lo podemos explicar más claro si decimos: "se declaran pero no se escribe código de lo que tiene que hacer". Esto se hace porque más adelante

© Alfaomega - RC Libros

tendremos la opción de colocarles código y decirles que funcionen de "X" manera para "Y" caso.

Para crear la interfaz vamos a ir al directorio *src* y pulsar el botón derecho, después vamos a seleccionar *New* y luego *Interface*, es lo mismo que hacemos cuando creamos una clase, pero en este caso lo que creamos es una interfaz y le colocamos el nombre de *EnviarMensaje.java*.

Camino: **Clic derecho directorio src -> New -> Interface(nombre EnviarMensaje.java)**

Después solamente declaramos un método *enviarDatos (String mensaje)*, es decir, que recibirá una cadena que lleva de nombre *mensaje*. El código es el siguiente:

```
package com.example.ejemplofragmentos;

public interface EnviarMensaje {
    public void enviarDatos(String mensaje);
}
```

Creando código de los fragmentos

Ahora vamos a crear la lógica de los fragmentos, para eso vamos al directorio *src* y pulsamos el botón derecho, seleccionamos *New* y después *Class*.

Camino: **Clic derecho directorio src -> New -> Class(nombre Derecha)**

Este es el código del archivo *Derecha.java*:

```
package com.example.ejemplofragmentos;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
```

© Alfaomega - RC Libros

```
    public class Derecha extends Fragment{
        ViewrootView;
        TextView txt;

        public ViewonCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState){

            rootView = inflater.inflate(R.layout.derecha,
container,false);
            txt = (TextView)rootView.findViewById(R.id.txt);
            return rootView;
        }

        public void ObtenerDatos(String mensaje){
            txt.setText(mensaje);
        }

    }
```

Declaramos la clase derecha y la hacemos herencia de la clase *Fragment*, declaramos 2 elementos que vamos a usar que son un *TextView* y *View*. La vista nos va a servir para indicar la interfaz gráfica del fragmento, esta es una forma de colocarle una "cara" al fragmento:

```
    public class Derecha extends Fragment{
        ViewrootView;
        TextView txt;
```

Indicaremos de dónde obtendremos la lista, que en el caso del fragmento *derecha* es *derecha.xml*. Además, enlazamos el *TextView* que declaramos anteriormente al elemento *TextView* del XML y finalmente devolvemos la vista que vamos a utilizar en la sentencia *return*:

```
        public ViewonCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState){
```


© Alfaomega - RC Libros

```
            rootView = inflater.inflate(R.layout.derecha,
container,false);
            txt = (TextView)rootView.findViewById(R.id.txt);
            return rootView;
        }
```

Comentamos anteriormente que el fragmento de la derecha tendrá un *TextView* y ese colocará un dato que recibiremos desde el fragmento *izquierda*, para esto creamos un método que llamaremos *ObtenerDato* y este se encargará de recibir una cadena de texto que se llama *"mensaje"* y colocarle al *TextView* este texto, el código es el siguiente:

```
    public void ObtenerDatos(String mensaje){
            txt.setText(mensaje);
        }
```

Creamos la clase que contiene el código del fragmento de la izquierda:

```
    package com.example.ejemplofragmentos;

    import android.app.Activity;
    import android.app.Fragment;
    import android.os.Bundle;
    import android.view.LayoutInflater;
    import android.view.View;
    import android.view.ViewOnClickListener;
    import android.view.ViewGroup;
    import android.widget.Button;
    import android.widget.EditText;

    public class Izquierda extends Fragment{
        ViewrootView;
        Button boton;
        EditText campo;
        EnviarMensaje EM;

        public ViewonCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState){
```

```
            rootView = inflater.inflate(R.layout.izquierda,
container,false);
            campo = (EditText) rootView.findViewById(R.id.campotxt);
            boton = (Button) rootView.findViewById(R.id.boton);
            boton.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {
                    // TODO Auto-generated method stub
                    String mensaje;
                    mensaje = campo.getText().toString();
                    EM.enviarDatos(mensaje);
                }
            });
            return rootView;
        }
        public void onAttach(Activity activity){
            super.onAttach(activity);
            try {
                EM = (EnviarMensaje) activity;
            } catch (ClassCastException e) {
                // TODO: handle exception
                throw new ClassCastException("Necesitas
implementar un msg");
            }
        }
    }
```

Para crear el archivo hacemos lo mismo que ya hicimos para *Derecha.java*. Declaramos la clase *Izquierda* y la hacemos herencia de *Fragment*. Declaramos los elementos *rootView*, *boton*, *EditText* y *EnviarMensaje*, este último es la interfaz que declaramos anteriormente:

```
    public class Izquierda extends Fragment{
        ViewrootView;
        Button boton;
        EditText campo;
        EnviarMensaje EM;
```


© Alfaomega - RC Libros

Ahora le indicamos que la interfaz es *R.layout.izquierda* y enlazamos los elementos del XML con los del *.java*, el *EditText* y el botón:

```
rootView = inflater.inflate(R.layout.izquierda, container,false);
        campo = (EditText) rootView.findViewById(R.id.campotxt);
        boton = (Button) rootView.findViewById(R.id.boton);
```

Cuando se presiona el botón le vamos a indicar que tome el texto del *EditText* y que lo coloque en una cadena de texto que se llama *"mensaje"*, este mismo se lo pasamos al elemento que declaramos de la interfaz que se llama *EM* y le pasamos este *"mensaje"* como parámetro. Antes de finalizar el *onCreateView* devolvemos la vista:

```
boton.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            String mensaje;
            mensaje = campo.getText().toString();
            EM.enviarDatos(mensaje);
        }
    });
    return rootView;
```

Aprovecharemos el método *onAttach* para definir la parte en donde realizaremos la comunicación utilizando también la clase *Activity* y el *EM*. Tendremos que agregar una *Exception* que nos solicitará Java para realizar este tipo de instrucciones, podemos colocarle cualquier mensaje para que nos avise de que en caso de lograrlo es probablemente porque no tenemos implementada la forma de enviar un mensaje; el código queda así:

```
public void onAttach(Activity activity){
    super.onAttach(activity);
    try {
        EM = (EnviarMensaje) activity;
    } catch (ClassCastException e) {
        // TODO: handle exception
```

```
                throw new ClassCastException("Necesitas
implementar un msg");
            }
        }
```

Finalmente vamos a nuestra actividad principal que tiene el siguiente código:

```
    public class MainActivity extends Activity implements
EnviarMensaje{

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


        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is
present.
            getMenuInflater().inflate(R.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();
            if (id == R.id.action_settings) {
                return true;
            }
            return super.onOptionsItemSelected(item);
        }


          @Override
          public void enviarDatos(String mensaje) {
```


© Alfaomega - RC Libros

```
            // TODO Auto-generated method stub
            Derecha derecha = (Derecha)
getFragmentManager().findFragmentById(R.id.derecha);
            derecha.ObtenerDatos(mensaje);
        }
    }
```

Declaramos la *MainActivity*, hacemos la herencia de *Activity* e implementamos *EnviarMensaje*, al implementarlo nos sucederá algo parecido al *OnClickListener*:

```
    public class MainActivity extends Activity implements
EnviarMensaje{
```

Nos comentará Eclipse que tenemos métodos sin implementar, le decimos que los implemente. Se va a crear el método *enviarDatos* que está esperando una cadena *"mensaje"*:

```
    public void enviarDatos(String mensaje) {
```

Dentro de este método declaramos un elemento *Derecha* y utilizamos el *getFragmentManager* para traerlo y utilizarlo:

```
    Derecha derecha = (Derecha)
getFragmentManager().findFragmentById(R.id.derecha);
```

Finalmente para pasarle el dato al fragmento *derecha*, utilizamos de "derecha" que acabamos de crear el método *ObtenerDatos()* y le pasamos el mensaje como parámetro.

```
    derecha.ObtenerDatos(mensaje);
```

Registrar los fragmentos en el manifest

Como las actividades que creamos anteriormente, los fragmentos también se deben registrar en el archivo Manifest de nuestra aplicación para poder funcionar, en nuestro ejemplo nuestro *manifest* debe tener el siguiente código:

```
    <?xml version="1.0" encoding="utf-8"?>
    <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.ejemplofragmentos"
        android:versionCode="1"
        android:versionName="1.0" >

        <uses-sdk
            android:minSdkVersion="14"
            android:targetSdkVersion="14" />

        <application
            android:allowBackup="true"
            android:icon="@drawable/ic_launcher"
            android:label="@string/app_name"
            android:theme="@style/AppTheme" >
            <activity
                android:name=".MainActivity"
                android:label="@string/app_name" >
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />

                    <category
android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
            <activity
                android:name="com.example.ejemplofragmentos.Izquierda"
                android:label="@string/app_name" >
                <intent-filter>
                    <action
android:name="android.intent.action.IZQUIERDA" />

                    <category
android:name="android.intent.category.DEFAULT" />
                </intent-filter>
            </activity>
             <activity
                android:name="com.example.ejemplofragmentos.Derecha"
                android:label="@string/app_name" >
```


© Alfaomega - RC Libros

```
            <intent-filter>
                <action android:name="android.intent.action.DERECHA"
/>

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

</manifest>
```

Esta es la ventana principal de nuestro ejemplo:



Cambiamos el texto, pulsamos el botón y el texto del fragmento *derecha* cambia como la imagen:



© Alfaomega - RC Libros

20 PÉRDIDA DE INFORMACIÓN

Algo que es recurrente cuando empezamos en Android es que tenemos algunas pérdidas de información, esto no se debe siempre a que hagamos las cosas mal, pero sí se debe a que no estamos entendiendo del todo el proceso de Android para manejar los datos, las actividades y otros elementos.

Ya hemos visto en otro capítulo cuál es el ciclo de vida de una actividad, en este capítulo vamos a ver primero cómo se pierden los datos habitualmente.

Vamos a generar una interfaz con 3 elementos, un TextView, un EditText y un botón. Este es el código del archivo XML:

```
<?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="datos.de.manejo.app.com.appmanejodedatos.MainActivity">

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

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

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


</LinearLayout>
```

En el primer bloque de código tenemos un LinearLayout que tiene una orientación vertical, así que todos los elementos que vayamos a colocar se irán posicionando uno debajo del otro, tenemos 3 elementos, cada uno con su respectivo id para poder acceder a él desde el código de Java.

Ahora vamos a ver cómo es la vista visualmente:



© Alfaomega - RC Libros

Vemos que el texto del textview es "Hello world!", el edittext está vacío en un principio y el botón solo contiene el texto "Enviar", esto es importante porque podemos observar cómo se pierden los datos de forma natural en Android.

Este es nuestro código del MainActivity.java:

```
package datos.de.manejo.app.com.appmanejodedatos;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   static final String TAG = "DATO";
   TextView textView;
   EditText editText;
   Button boton;



   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(TAG, "oncreate");
       textview = (TextView) findViewById(R.id.textview);
       editText = (EditText) findViewById(R.id.editText);
       boton = (Button) findViewById(R.id.boton);

      botón.setOnClickListener(new View.OnClickListener){
          @Override
          public void onClick(View v){
                    textView.setText(editText.getText().toString());
          }
```

```
    }

}


@Override
protected void onStart() {
    super.onStart();
    Log.d(TAG, "onstart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "onresume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(TAG, "onpause");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(TAG, "onrestart");
}

@Override
protected void onDestroy() {
```


© Alfaomega - RC Libros

```
    super.onDestroy();
    Log.d(TAG, "ondestroy");
  }
}
```

Podemos ver en el código que hacemos la definición de una clase y declaramos los elementos que tenemos en el XML y nos interesa utilizar tres, agregamos una cadena de texto que nos va a servir para colocarla como mensaje TAG en los envíos de mensajes a la consola que vamos a utilizar:

```
  public class MainActivity extends AppCompatActivity {
 static final String TAG = "DATO";
 TextView textView;
 EditText editText;
 Button boton;
```

Reescribimos el método onCreate, le indicamos al método onCreate que lo primero que ejecute sea enviar un mensaje a la consola que diga que estamos en "oncreate" y así podemos ver cuántas veces pasamos por este método, asignamos la referencia de los elementos del código Java con los del archivo XML:

```
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(TAG, "oncreate");
     textview = (TextView) findViewById(R.id.textview);
     editText = (EditText) findViewById(R.id.editText);
     botón = (Button) findvViewById(R.id.boton);
```

Definimos el onClickListener y el onClick para el botón en donde simplemente le cambiamos el texto que tiene por defecto el TextView por el que contiene el EditText:

```
    botón.setOnClickListener(new View.OnClickListener){
        @Override
        public void onClick(View v){
                textView.setText(editText.getText().toString());
        }
}
```

Las últimas líneas de código, que es la mayor parte del código, son solamente para mandar un mensaje por consola cuando se ejecutan los métodos que sabemos tienen una actividad dentro de su ciclo de vida, la idea es que el lector pueda tener una idea más visual de todo lo que sucede durante el proceso de ejecución de esta aplicación:

```
    @Override
protected void onStart() {
    super.onStart();
    Log.d(TAG, "onstart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "onresume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(TAG, "onpause");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
}

@Override
protected void onRestart() {
    super.onRestart();
```


© Alfaomega - RC Libros

```
    Log.d(TAG, "onrestart");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "ondestroy");
}
```

Ya que colocamos texto en el editText podemos ver cómo al pulsar el botón, el texto que tiene el elemento TextView cambia y ahora se ve el que teníamos en el elemento editText:



Como vemos todo funciona bien, pero el detalle es que, si cambiamos de posición el dispositivo y lo hacemos rotar, la actividad se destruye y se crea de nueva cuenta:



Podemos observar que se perdió lo que ya se había hecho en el textview, esa es la pérdida de datos más común en las aplicaciones Android.

Evitar pérdida de datos

Existen varias formas, al menos tres que son las comunes para evitar la pérdida de datos, pero algunas son más complejas que otras y no siempre nos vienen bien las formas, depende en realidad de los datos que manejemos y la complejidad de la activity. En este caso lo sencillo es hacer lo siguiente:

Dentro de las actividades tendremos dos métodos:

- onSaveInstanceState
- onRestoreInstanceState

El primer método nos permite almacenar el estado de la actividad y por lo tanto también algunos datos, aunque sigue siendo nuestra la responsabilidad de indicar cuáles son los datos que vamos a almacenar, por lo tanto, nosotros tendremos que estar pendientes de las variables, los tipos de datos y de todo lo que sabemos que necesitaremos en caso de destruir la actividad y posible pérdida de información.

El segundo método nos permite recuperar la información en caso de que se tenga que recuperar, si no se tiene nada guardado simplemente no pasará nada, pero en caso de que se tenga algo guardado, lo recuperará y en ese momento se tendrán que considerar los datos e información para acomodar en la actividad o usar donde corresponda.

Así tendríamos que escribir el método onSaveInstanceState:

© Alfaomega - RC Libros

```
@Override
public void onSaveInstanceState(Bundle outState ) {
   outState.putString("guardado",textView.getText().toString());
   super.onSaveInstanceState(outState);


}
```

Podemos ver que el método utiliza la clase Bundle para poder mantener los datos, simplemente tenemos que saber el tipo de datos que queremos almacenar y "empujarlo" al bundle, la instrucción en este caso que permite hacer eso es putString y le pasamos el dato con el nombre de "guardado" y el valor, este valor lo obtenemos del textview, así que lo último que tenía el textview es lo que se mantiene.

Este es el código del onRestoreInstanceState:

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   textView.setText(savedInstanceState.getString("guardado"));
}
```

En este código vemos que lo que se hace es después del *super* es asignarle un texto al textview, este texto se recupera de la instancia si es que existe una instancia guardada y ver si se puede recuperar, lo importante de esta instrucción es que la instrucción tenga el nombre correcto del dato, es decir, que deben de coincidir los nombres de lo que se guardó con lo que se recupera, en caso contrario el lector podría pensar que no está funcionando pero es común que el error sea de dedo y no de la aplicación:

Ahora veamos cómo queda el código completo de nuestra MainActivity:

```
package datos.de.manejo.app.com.appmanejodedatos;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    static final String TAG = "DATO";
    TextView textView;
    EditText editText;
    Button boton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "oncreate");

        textView = (TextView) findViewById(R.id.textview);
        editText = (EditText) findViewById(R.id.edittext);
        boton = (Button) findViewById(R.id.boton);

        boton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                textView.setText(editText.getText().toString());

            }
        });

    }

    @Override
    public void onSaveInstanceState(Bundle outState ) {
        outState.putString("guardado",textView.getText().toString());
        super.onSaveInstanceState(outState);

    }

    @Override
```


© Alfaomega - RC Libros

```
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    textView.setText(savedInstanceState.getString("guardado"));
}

@Override
protected void onStart() {
    super.onStart();
    Log.d(TAG, "onstart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "onresume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(TAG, "onpause");
}

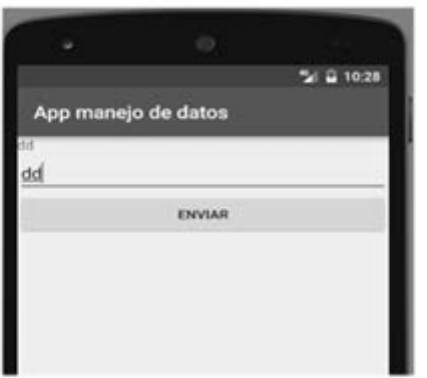
@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(TAG, "onrestart");
}

@Override
protected void onDestroy() {
```

```
        super.onDestroy();
        Log.d(TAG, "ondestroy");
    }
}
```

Esta es la imagen cuando pulsamos el botón y asignamos el texto al textview:



Ahora vemos la imagen ya que hicimos la rotación del teléfono:



Podemos observar que los datos se mantienen, en realidad no es que se mantengan, simplemente salvamos el estado del textview y lo recuperamos en la nueva actividad, el tema del manejo de información de Android es bastante interesante, este tema se recomienda que el lector lo aborde con profundidad ya que es una de las principales necesidades en el mundo Android para una aplicación con calidad.

© Alfaomega - RC Libros

21 LISTVIEW

El Listview tiene dos formas de usarse, la forma estándar y la personalizada, en este capitulo crearemos uno personalizado, el clásico es de color negro y con una línea de texto en cada elemento, vamos a crear uno que tenga:

- Una imagen
- Texto por cada ítem
- Color diferente
- Encabezado
- Al hacer clic en un elemento nos va a mostrar el texto del elemento al que le hicimos clic

Tendremos los siguientes archivos:

- MainActivity.java
- listview_item_row.xml
- list_header_row.xml
- FrutasVerdurasAdapter.java
- FrutasVerduras.java
- Activity_main.xml

Cada archivo tiene su función:

- Diseño de los ítems de la lista. (Archivo listview_item_row.xml)
- Diseño del encabezado de la lista. *Opcional (archivo list_header_row.xml)

- 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" >
```


© Alfaomega - RC Libros

```
    <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;
    }
}
```


© Alfaomega - RC Libros

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;
```

© Alfaomega - RC Libros

```
    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 ViewgetView(int position, View convertView,
ViewGroup parent) {

            ViewconvertView;
            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);
```


© Alfaomega - RC Libros

```
            }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 *LayoutInflaterrow,* que es lo que se muestra al finalizar todo. El código es el siguiente:

```
    public ViewgetView(int position, View convertView, ViewGroup
parent) {

            ViewconvertView;
            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;
        }
```


© Alfaomega - RC Libros

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();
            }
```


© Alfaomega - RC Libros

```
            });
    }


    @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:

© Alfaomega - RC Libros

```
        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:

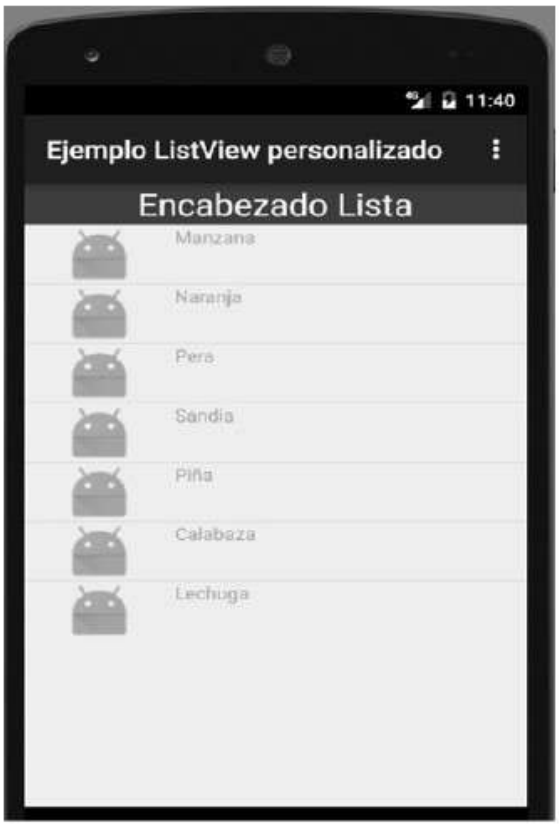
```
        lv= (ListView) findViewById(R.id.lv);
        ViewgetLayoutInflater().inflate(R.layout.list_header_row, null);
        lv.addHeaderView(header);
        lv.setAdapter(adapter);
```

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:

© Alfaomega - RC Libros



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



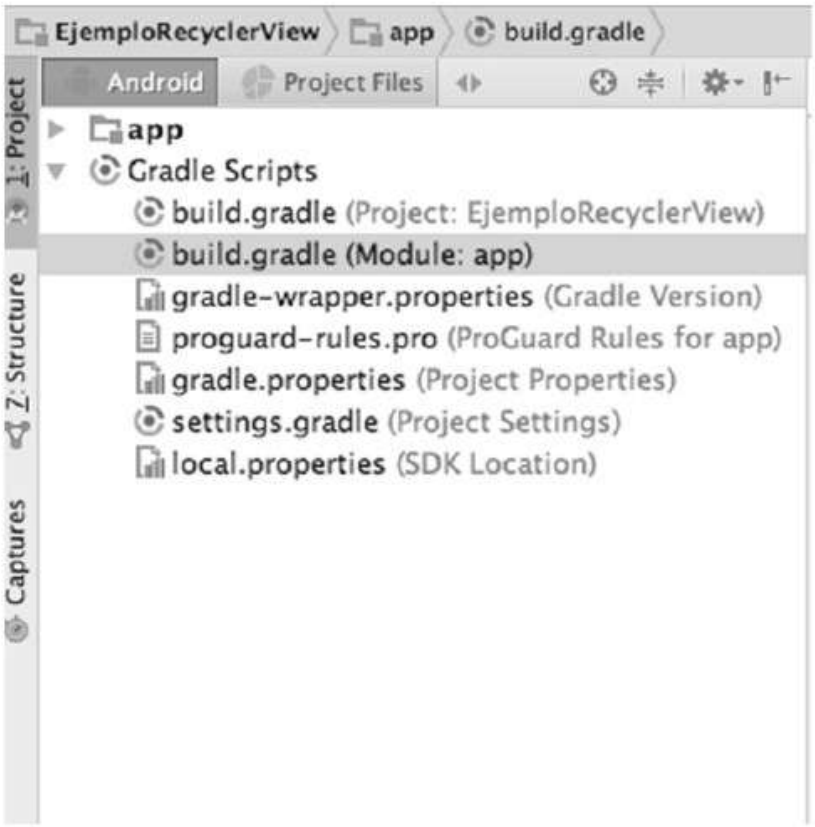
22 RECYCLERVIEW

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:

© Alfaomega - RC Libros

```
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 version 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">
```

© Alfaomega - RC Libros

```
    <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">
```


© Alfaomega - RC Libros

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 alto 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 RecyclerAdapter(Data);

        recyclerView.setAdapter(adapter);
```


© Alfaomega - RC Libros

```
}

@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;
```

© Alfaomega - RC Libros

```
        }


        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"};
```


© Alfaomega - RC Libros

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 RecyclerAdapter(Data)
```

Este adaptador lo tenemos definido en una clase diferente que la llamamos RecyclerAdapter, 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 RecylerAdapter:

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

        private String[] myData;



        public RecyclerAdapter(String[] Data){

        myData = Data;

        }

        @Override

        public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
```


© Alfaomega - RC Libros

```
        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 RecyclerAdapter extends
RecyclerView.Adapter<RecyclerAdapter.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 RecyclerAdapter(String[] Data){

    myData = Data;

}
```


© Alfaomega - RC Libros

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>
```


© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

```
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 RecyclerAdapter(Data)
```


© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

23 CARDVIEW

El CardView es un elemento nuevo en Android, el elemento RecyclerView del capítulo anterior y el elemento cardview que veremos en este 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"
```


© Alfaomega - RC Libros

```
            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;
```


© Alfaomega - RC Libros

```
   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 RecyclerAdapter(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);
    }

}
```


© Alfaomega - RC Libros

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

```
public class RecyclerAdapter extends
RecyclerView.Adapter<RecyclerAdapter.ViewHolder> {
   private String[] myData;


   public RecyclerAdapter(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);


     }
  }
}
```


© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

24 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">
```


© Alfaomega - RC Libros

```
<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.

© Alfaomega - RC Libros

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 imagines 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.

© Alfaomega - RC Libros

```
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 RecyclerAdapter(Data);
        recyclerView.setAdapter(adapter);
         fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        }
    });
}

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


© Alfaomega - RC Libros

```
        // 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);
```

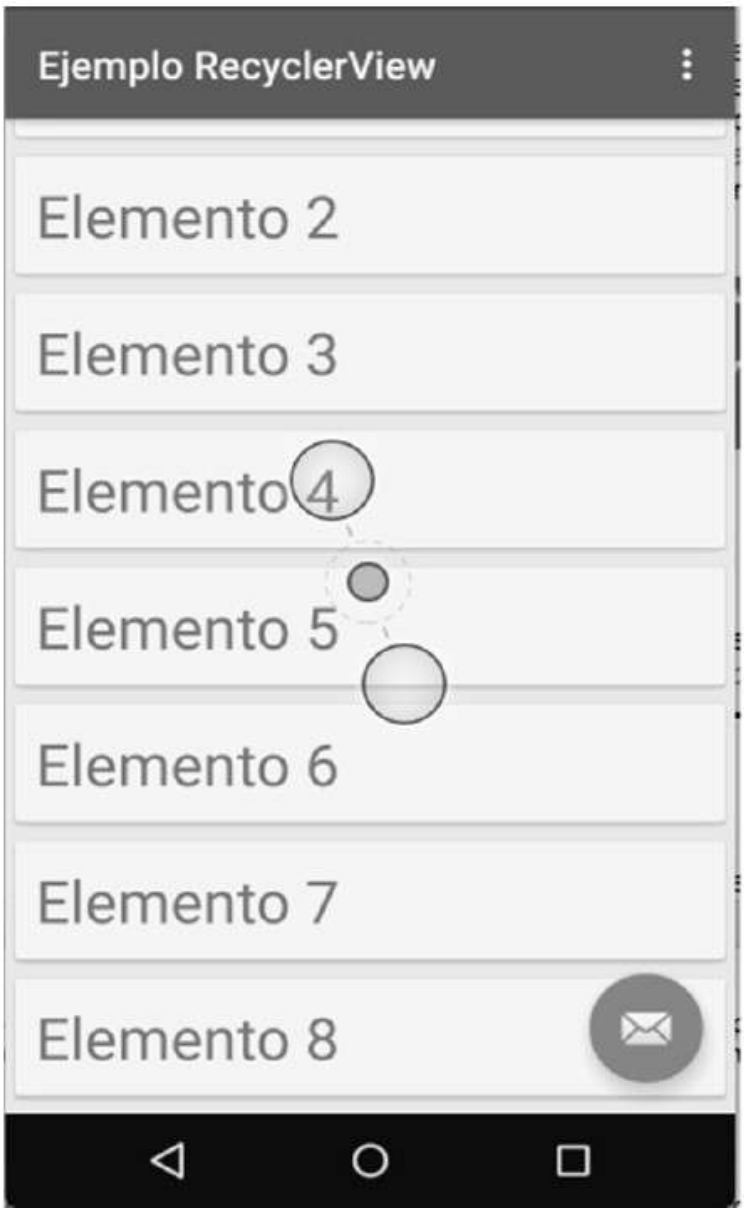
Para finalizar la implementación del elemento ahora vamos a definir el onClickListener como si fuera un botón común y corriente, dentro del OnClick puedes colocar las instrucciones que te interese ejecutar.

```
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:

© Alfaomega - RC Libros



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.

© Alfaomega - RC Libros

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 creemos 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",
```


© Alfaomega - RC Libros

```
                    "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 RecyclerAdapter(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) {
```


© Alfaomega - RC Libros

```
            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:

© Alfaomega - RC Libros



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:

© Alfaomega - RC Libros

```
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));
```

© Alfaomega - RC Libros

```
            adapter = new RecyclerAdapter(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) {
```


© Alfaomega - RC Libros

```
        // 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);
    }

}
```

© Alfaomega - RC Libros

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) {
```


© Alfaomega - RC Libros

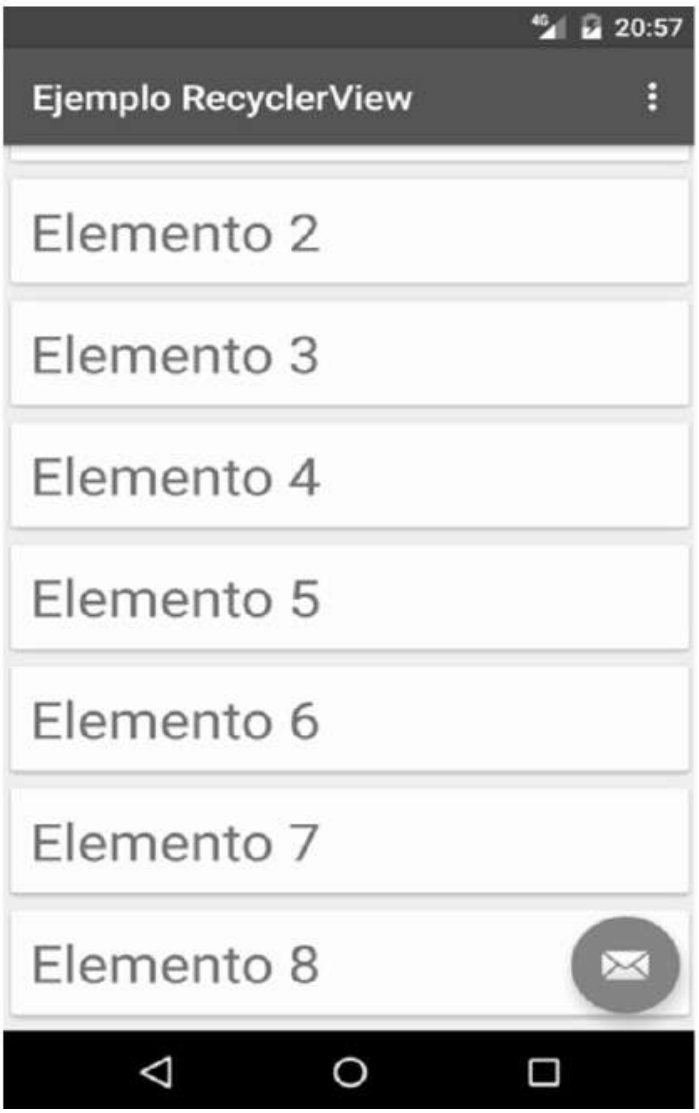
```
            finish();

        }

    };
```

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.

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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.

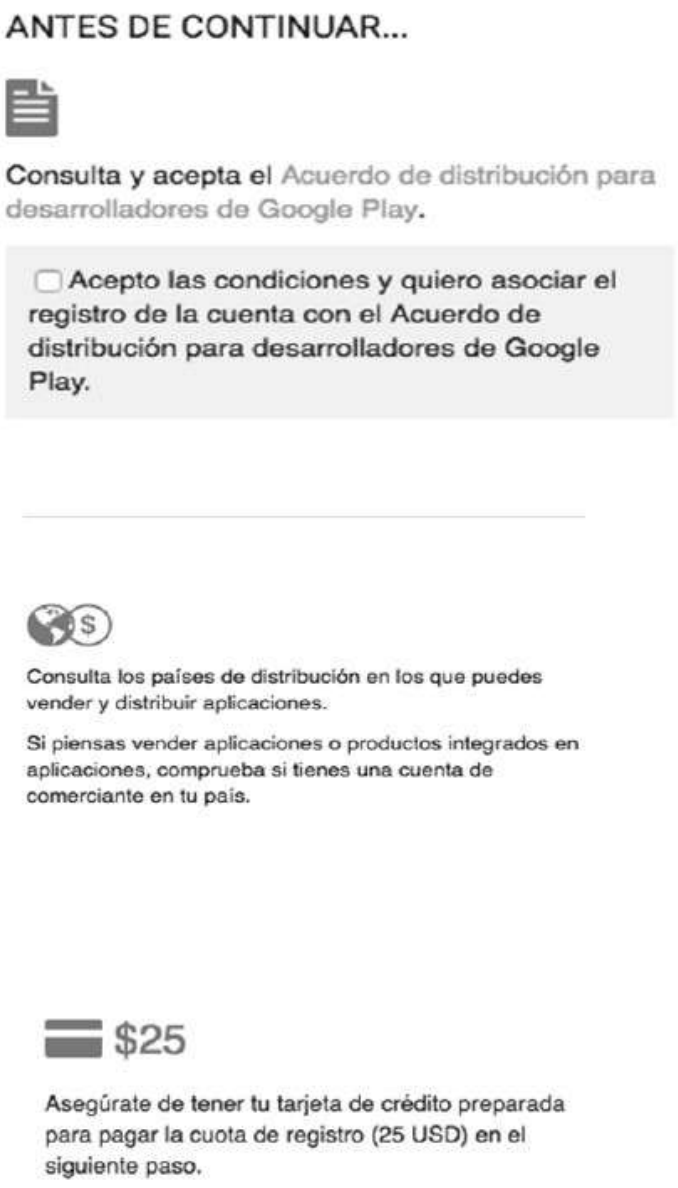
Comenzaremos diciendo que se necesita registrar o darse de alta como desarrollador, lo primero es ir al navegador y abrir Google, escribimos play store console, nos aparecen una serie de enlaces, el primero nos mandará a la consola de la tienda de desarrollo, en esa consola es donde administramos y configuramos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



© Alfaomega - RC Libros

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

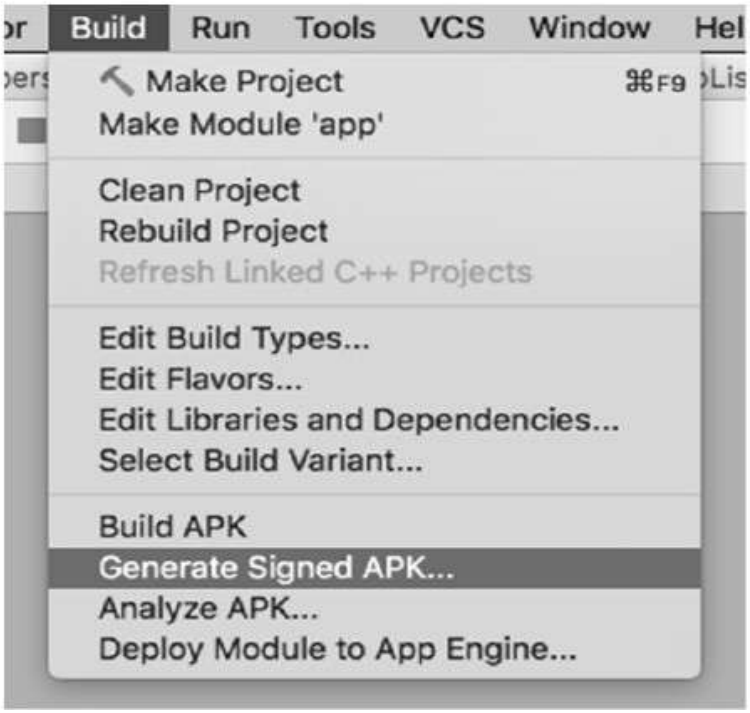
Generar APK

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:

© Alfaomega - RC Libros



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.

© Alfaomega - RC Libros



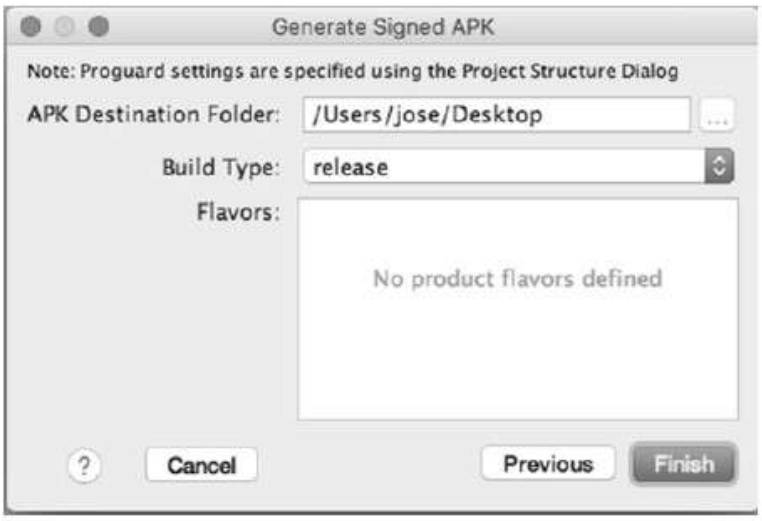
Al pulsar en “OK“ veremos la ventana anterior cargada de los datos, es muy importante que esa llave se guarde en un lugar donde se recuerde y se tenga de nuevo acceso, en el caso de que nuestra aplicación ya esté publicada y queramos actualizarla, la tienda de aplicaciones nos va a solicitar la llave.

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:

© Alfaomega - RC Libros



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.

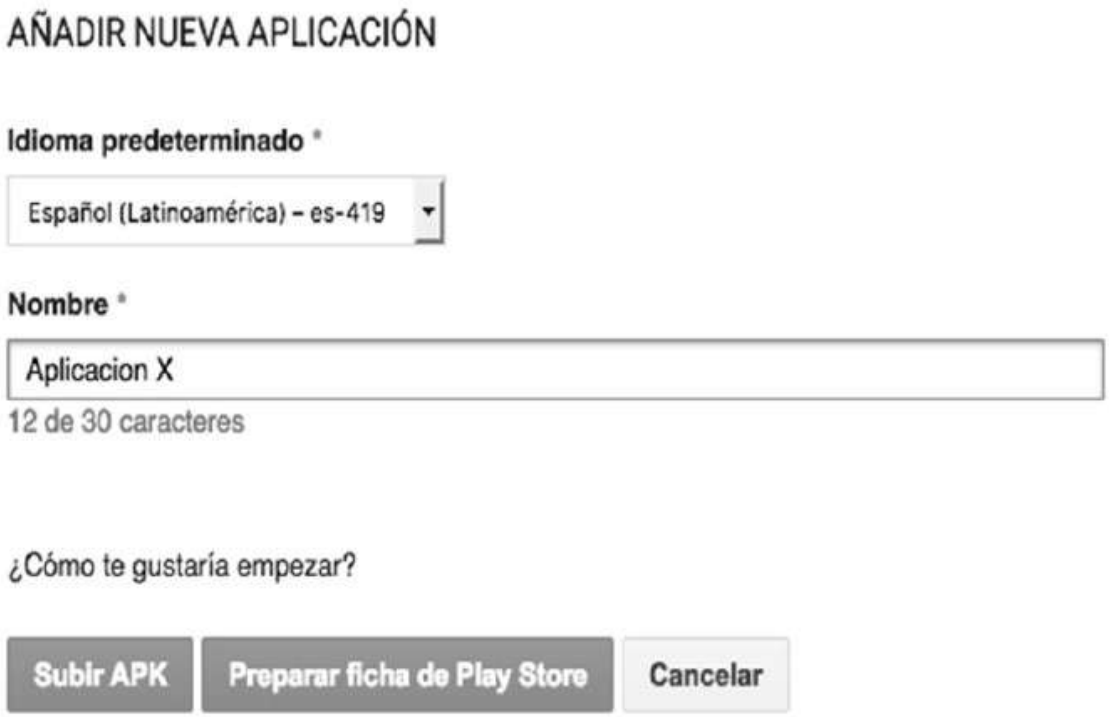
Subir APK a la Play Store

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:



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.



© Alfaomega - RC Libros

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:



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:

© Alfaomega - RC Libros



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 destaca
Inglés (Estados U
1024 x 500
Archivo JPG o PN



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.



© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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:



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

© Alfaomega - RC Libros

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.

© Alfaomega - RC Libros

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

Pixeles, 107, 109
Pulgada, 108
Puntos, 109

R

raw, 196
RecyclerView, 251

S

Sensores, 203
SnackBar, 289
String, 133

T

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

W

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