



Contenido

Audiencia	5
Prerrequisitos	5
Descripción general	5
¿Qué es Android?	5
¿Por qué Android?	5
Características de Android.....	6
Aplicaciones de Android.....	7
Categorías de aplicaciones de Android	8
Historia de Android.....	8
¿Qué es el nivel de API?.....	9
Configuración del entorno	10
Configurar el kit de desarrollo de Java (JDK)	11
IDE de Android.....	11
• Arquitectura	12
Kernel de Linux.....	12
Bibliotecas	12
Bibliotecas de Android.....	13
Tiempo de ejecución de Android.....	13
Marco de aplicación	14
Aplicaciones	14
Componentes de la aplicación	14
Activities.....	15
Services	15
Broadcast Receivers.....	16
Content Providers.....	16
Componentes adicionales.....	16



Ejemplo de Hello World	17
Crear aplicación de Android.....	17
Anatomía de la aplicación de Android.....	20
El archivo de actividad principal	22
El archivo manifest	22
El archivo strings	24
El archivo layout	24
Ejecutando la Aplicación	25
Organización y acceso a los recursos de Android	26
Organizar recursos en Android Studio	26
Recursos alternativos	28
Acceso a recursos	29
Acceso a recursos en código.....	29
Ejemplo	29
Ejemplo	30
Ejemplo	30
Acceso a recursos en XML	30
Actividades	31
Ejemplo	33
Servicios.....	36
Ejemplo	40
Receptores de difusión	45
Creación del receptor de transmisión.....	45
Registro de receptor de transmisión.....	46
Difusión de intenciones personalizadas	47
Ejemplo	48
Proveedores de contenido	52
URI de contenido	53
Crear proveedor de contenido	53
Ejemplo	54
Fragmentos	65
Ciclo de vida del fragmento	66
¿Cómo usar Fragments?	68
Tipos de fragmentos	68

Fragmentos individuales	68
Ejemplo	69
Fragmento de lista.....	75
Ejemplo	76
Ejecutando la Aplicación	80
Transición de fragmentos.....	81
¿Qué es una transición?	81
Ejemplo	82
Ejecutando la Aplicación	88
Intents and Filters	90
Objetos intent	92
Action	92
Data	92
Categoría	94
Extras	94
Flags	94
ComponentName	95
Tipos de intent	95
Intenciones explícitas	95
Intenciones implícitas	96
Ejemplo	97
Intent filters	102
Ejemplo	103
Uso de interfaces.....	109
diseños de interfaz de usuario.....	109
Tipos de diseño de Android	111
Atributos de diseño	112
Ver identificación	115
Diseño lineal de Android	115
Atributos de LinearLayout	115
Ejemplo	116
Relative Layout	119
Atributos de diseño relativo	120
Ejemplo	123

Table Layout.....	126
Atributos de TableLayout	126
Ejemplo	127
Absolute Layout	130
Atributos de AbsoluteLayout	131
Constructores públicos	131
Ejemplo	132
Frame Layout	134
Atributos de FrameLayout	135
Ejemplo	136
ListView	138
Atributos de ListView	139
ArrayAdapter	140
Ejemplo	141
SimpleCursorAdapter	144
Grid View	145
Atributos de GridView	146
Ejemplo	147
Ejemplo de subactividad	151
Controles de la interfaz de usuario	155
Controles de IU de Android	156
Crear controles de IU	158
Control TextView	158
Atributos de TextView	158
Ejemplo	161
Ejercicio	164

Android es un sistema operativo de código abierto y basado en Linux para dispositivos móviles como teléfonos inteligentes y tabletas. Android fue desarrollado por Open Handset Alliance, liderada por Google y otras empresas. Este tutorial le enseñará la programación básica de Android y también lo llevará a través de algunos conceptos avanzados relacionados con el desarrollo de aplicaciones de Android.

Audiencia

Este tutorial ha sido preparado para principiantes para ayudarlos a comprender la programación básica de Android. Después de completar este tutorial, se encontrará con un nivel moderado de experiencia en la programación de Android, desde donde puede llevarse a los siguientes niveles.

Prerrequisitos

La programación de Android se basa en el lenguaje de programación Java, por lo que si tiene conocimientos básicos de programación Java, será divertido aprender a desarrollar aplicaciones de Android.

Descripción general

¿Qué es Android?

Android es un sistema **operativo de** código abierto y basado en Linux para dispositivos móviles como teléfonos inteligentes y tabletas. Android fue desarrollado por *Open Handset Alliance*, liderada por Google y otras empresas.

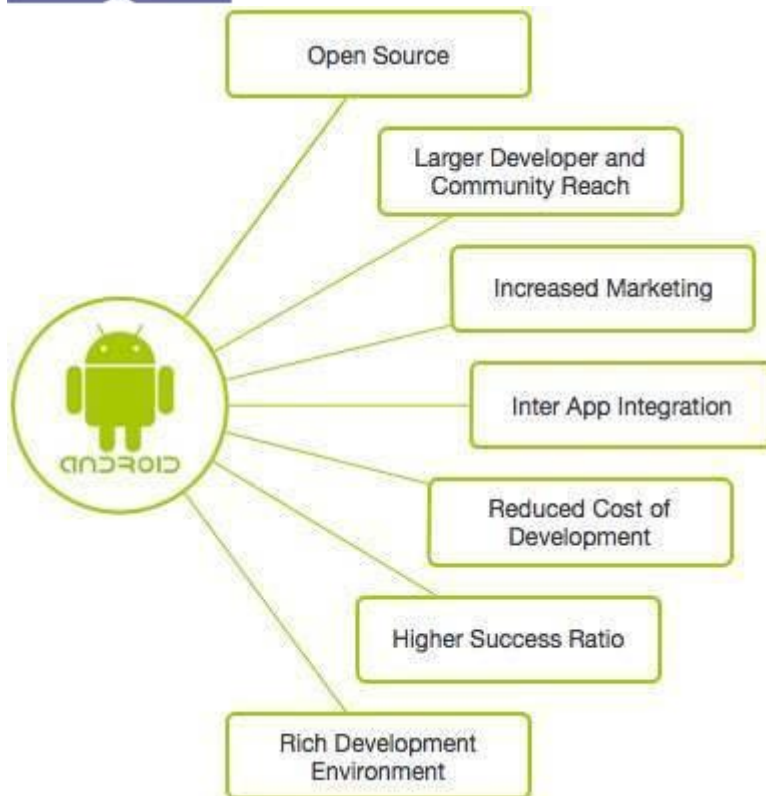
Android ofrece un enfoque unificado para el desarrollo de aplicaciones para dispositivos móviles, lo que significa que los desarrolladores solo necesitan desarrollar para Android, y sus aplicaciones deberían poder ejecutarse en diferentes dispositivos con tecnología Android.

La primera versión beta del kit de desarrollo de software de Android (SDK) fue lanzada por Google en 2007, mientras que la primera versión comercial, Android 1.0, se lanzó en septiembre de 2008.

El 27 de junio de 2012, en la conferencia Google I / O, Google anunció la próxima versión de Android, 4.1 **Jelly Bean**. Jelly Bean es una actualización incremental, con el objetivo principal de mejorar la interfaz de usuario, tanto en términos de funcionalidad como de rendimiento.

El código fuente para Android está disponible con licencias de software gratuitas y de código abierto. Google publica la mayor parte del código bajo la licencia Apache versión 2.0 y el resto, cambios en el kernel de Linux, bajo la licencia pública general GNU versión 2.

¿Por qué Android?



Características de Android

Android es un potente sistema operativo que compite con Apple 4GS y admite excelentes funciones. Algunos de ellos se enumeran a continuación:

No Señor.	Característica y descripción
1	Hermosa interfaz de usuario La pantalla básica del sistema operativo Android proporciona una interfaz de usuario hermosa e intuitiva.
2	Conectividad GSM / EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC y WiMAX.
3	Almacenamiento SQLite, una base de datos relacional liviana, se utiliza para fines de almacenamiento de datos.
4	Soporte de medios



H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

5

Mensajería

SMS y MMS

6

navegador web

Basado en el motor de diseño WebKit de código abierto, junto con el motor JavaScript V8 de Chrome compatible con HTML5 y CSS3.

7

Multitáctil

Android tiene soporte nativo para multitáctil que inicialmente estaba disponible en teléfonos como el HTC Hero.

8

Multitarea

El usuario puede saltar de una tarea a otra y, al mismo tiempo, se pueden ejecutar varias aplicaciones simultáneamente.

9

Widgets redimensionables

Los widgets son redimensionables, por lo que los usuarios pueden expandirlos para mostrar más contenido o reducirlos para ahorrar espacio.

10

Multi lenguaje

Admite texto de una sola dirección y bidireccional.

11

GCM

Google Cloud Messaging (GCM) es un servicio que permite a los desarrolladores enviar datos de mensajes cortos a sus usuarios en dispositivos Android, sin necesidad de una solución de sincronización patentada.

12

Wi-Fi directo

Una tecnología que permite que las aplicaciones descubran y se emparejen directamente, a través de una conexión de igual a igual de gran ancho de banda.

13

Android Beam

Una tecnología popular basada en NFC que permite a los usuarios compartir instantáneamente, simplemente tocando dos teléfonos habilitados para NFC.

Aplicaciones de Android

Las aplicaciones de Android generalmente se desarrollan en el lenguaje Java utilizando el kit de desarrollo de software de Android.

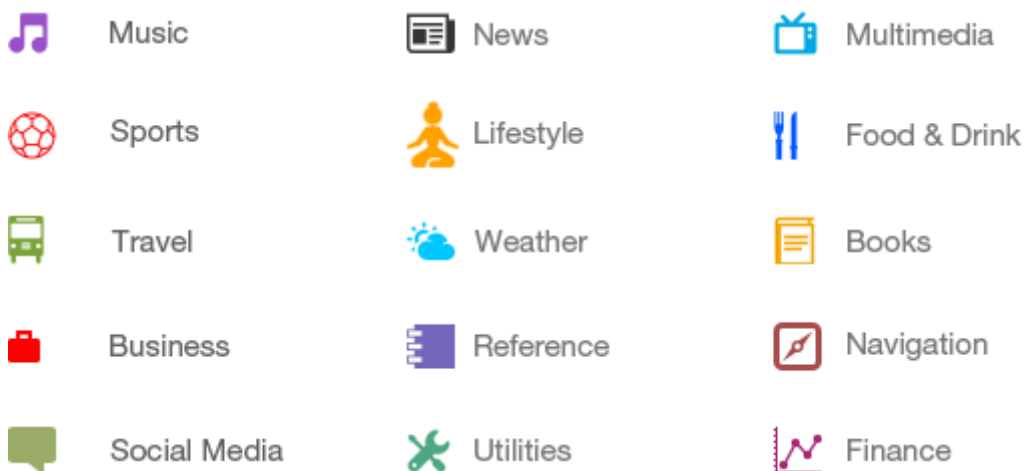
Una vez desarrolladas, las aplicaciones de Android se pueden empaquetar fácilmente y agotar a través de una tienda como **Google Play** , **SlideME** , **Opera Mobile Store** , **Mobango** , **F-droid** y **Amazon Appstore** .

Android funciona con cientos de millones de dispositivos móviles en más de 190 países de todo el mundo. Es la base instalada más grande de cualquier plataforma móvil y está creciendo rápidamente. Cada día se activan más de 1 millón de nuevos dispositivos Android en todo el mundo.

Este tutorial ha sido escrito con el objetivo de enseñarle cómo desarrollar y empaquetar aplicaciones de Android. Comenzaremos desde la configuración del entorno para la programación de aplicaciones de Android y luego profundizaremos para examinar varios aspectos de las aplicaciones de Android.

Categorías de aplicaciones de Android

Hay muchas aplicaciones de Android en el mercado. Las categorías principales son:



Historia de Android

Los nombres en clave de Android van de la A a la N actualmente, como Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop y Marshmallow. Entendamos la historia de Android en una secuencia.



**INSTITUTO TÉCNICO
DE ESTUDIOS
PROFESIONALES**



¿Qué es el nivel de API?

El nivel de API es un valor entero que identifica de forma única la revisión de la API del marco que ofrece una versión de la plataforma Android.

Versión de la plataforma	Nivel API	VERSION_CODE	
Android 6.0	23	MALVAVISCO	
Android 5.1	22	LOLLIPOP_MR1	
Android 5.0	21	CHUPETE	
Android 4.4W	20	KITKAT_WATCH	KitKat solo para wearables
Android 4.4	19	KIT KAT	
Android 4.3	18	JELLY_BEAN_MR2	
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	
Android 4.1, 4.1.1	dieciséis	FRIJOL DE JALEA	



Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	PANAL
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	PAN DE JENGIBRE
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	ROSQUILLA
Android 1.5	3	MAGDALENA
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

Configuración del entorno

Le alegrará saber que puede iniciar el desarrollo de su aplicación Android en cualquiera de los siguientes sistemas operativos:

- Microsoft Windows XP o una versión posterior.
- Mac OS X 10.5.8 o versión posterior con chip Intel.
- Linux, incluida la biblioteca GNU C 2.7 o posterior.

El segundo punto es que todas las herramientas necesarias para desarrollar aplicaciones de Android están disponibles gratuitamente y pueden descargarse de la Web. A continuación se muestra la lista de software que necesitará antes de comenzar la programación de la aplicación de Android.

- Java JDK5 o versión posterior
- Estudio de Android

Aquí los dos últimos componentes son opcionales y si está trabajando en una máquina con Windows, estos componentes le facilitan la vida mientras realiza el desarrollo de aplicaciones basadas en Java. Así que echemos un vistazo a cómo proceder para establecer el entorno requerido.

Configurar el kit de desarrollo de Java (JDK)

Puede descargar la última versión de Java JDK desde el sitio de Java de Oracle - Descargas de Java SE . Encontrará instrucciones para instalar JDK en archivos descargados, siga las instrucciones dadas para instalar y configurar la instalación. Finalmente, configure las variables de entorno PATH y JAVA_HOME para hacer referencia al directorio que contiene **java** y **javac** , normalmente `java_install_dir / bin` y `java_install_dir` respectivamente.

Si está ejecutando Windows e instaló el JDK en `C: \jdk1.8.0_102`, tendría que poner la siguiente línea en su archivo `C: \autoexec.bat`.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternativamente, también puede hacer clic con el botón derecho en *Mi PC* , seleccionar *Propiedades* , luego *Avanzado* , luego *Variables de entorno* . Luego, actualizaría el valor PATH y presionaría el botón OK.

En Linux, si el SDK está instalado en `/usr/local/jdk1.8.0_102` y usa el shell C, debe colocar el siguiente código en su archivo `.cshrc` .

```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

Alternativamente, si usa Android Studio, sabrá automáticamente dónde ha instalado su Java.

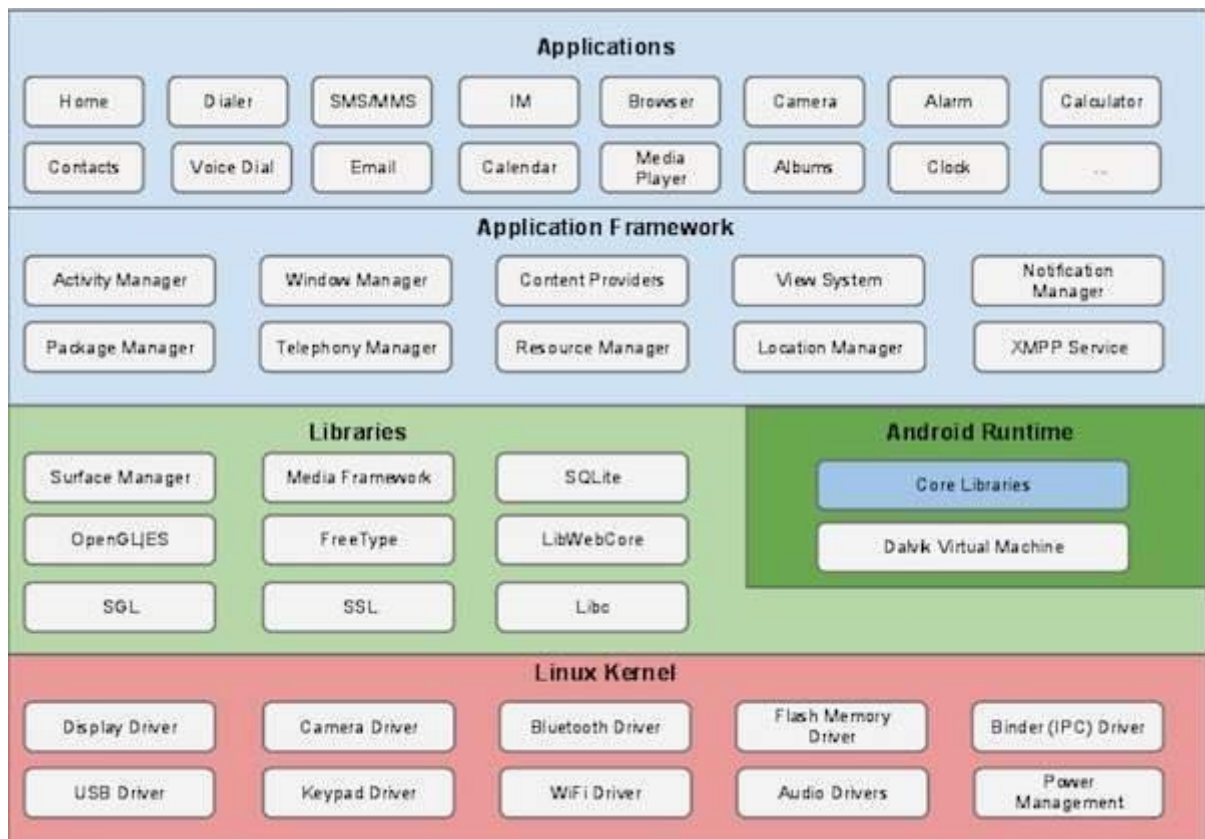
IDE de Android

Hay tantas tecnologías sofisticadas disponibles para desarrollar aplicaciones de Android, las tecnologías familiares, que utilizan predominantemente las siguientes herramientas

- Estudio de Android
- Eclipse IDE (obsoleto)

• Arquitectura

El sistema operativo Android es una pila de componentes de software que se divide aproximadamente en cinco secciones y cuatro capas principales, como se muestra a continuación en el diagrama de arquitectura.



Kernel de Linux

En la parte inferior de las capas está Linux - Linux 3.6 con aproximadamente 115 parches. Esto proporciona un nivel de abstracción entre el hardware del dispositivo y contiene todos los controladores de hardware esenciales como la cámara, el teclado, la pantalla, etc. Además, el kernel maneja todas las cosas en las que Linux es realmente bueno, como las redes y una amplia gama de controladores de dispositivos, que eliminan la molestia de interactuar con el hardware periférico.

Bibliotecas

Además del kernel de Linux, hay un conjunto de bibliotecas que incluyen el motor de navegador web de código abierto WebKit, una biblioteca de bibliotecas muy conocida, una base de datos SQLite que es un repositorio útil para almacenar y compartir datos de aplicaciones, bibliotecas para reproducir y

grabar audio y video, SSL bibliotecas responsables de la seguridad de Internet, etc.

Bibliotecas de Android

Esta categoría abarca las bibliotecas basadas en Java que son específicas del desarrollo de Android. Entre los ejemplos de bibliotecas de esta categoría se incluyen las bibliotecas del marco de aplicación, además de las que facilitan la creación de interfaces de usuario, el dibujo de gráficos y el acceso a bases de datos. Un resumen de algunas bibliotecas clave de Android disponibles para el desarrollador de Android es el siguiente:

- **android.app** : proporciona acceso al modelo de la aplicación y es la piedra angular de todas las aplicaciones de Android.
- **android.content** : facilita el acceso al contenido, la publicación y la mensajería entre aplicaciones y componentes de la aplicación.
- **android.database** : se utiliza para acceder a datos publicados por proveedores de contenido e incluye clases de administración de bases de datos SQLite.
- **android.opengl** : una interfaz Java para la API de representación de gráficos 3D de OpenGL ES.
- **android.os** : proporciona a las aplicaciones acceso a los servicios estándar del sistema operativo, incluidos mensajes, servicios del sistema y comunicación entre procesos.
- **android.text** : se utiliza para representar y manipular texto en la pantalla de un dispositivo.
- **android.view** : los bloques de construcción fundamentales de las interfaces de usuario de aplicaciones.
- **android.widget** : una rica colección de componentes de interfaz de usuario prediseñados, como botones, etiquetas, vistas de lista, administradores de diseño, botones de opción, etc.
- **android.webkit** : un conjunto de clases destinadas a permitir que las capacidades de navegación web se integren en las aplicaciones.

Habiendo cubierto las bibliotecas centrales basadas en Java en el tiempo de ejecución de Android, ahora es el momento de centrar nuestra atención en las bibliotecas basadas en C / C ++ contenidas en esta capa de la pila de software de Android.

Tiempo de ejecución de Android

Esta es la tercera sección de la arquitectura y está disponible en la segunda capa desde la parte inferior. Esta sección proporciona un componente clave

llamado **Dalvik Virtual Machine**, que es una especie de Java Virtual Machine especialmente diseñada y optimizada para Android.

La máquina virtual Dalvik hace uso de las funciones principales de Linux, como la gestión de memoria y el subproceso múltiple, que es intrínseco al lenguaje Java. Dalvik VM permite que cada aplicación de Android se ejecute en su propio proceso, con su propia instancia de la máquina virtual Dalvik.

El tiempo de ejecución de Android también proporciona un conjunto de bibliotecas centrales que permiten a los desarrolladores de aplicaciones de Android escribir aplicaciones de Android utilizando el lenguaje de programación estándar de Java.

Marco de aplicación

La capa Application Framework proporciona muchos servicios de nivel superior a las aplicaciones en forma de clases Java. Los desarrolladores de aplicaciones pueden hacer uso de estos servicios en sus aplicaciones.

El marco de Android incluye los siguientes servicios clave:

- **Administrador de actividades** : controla todos los aspectos del ciclo de vida de la aplicación y la pila de actividades.
- **Proveedores de contenido** : permite que las aplicaciones publiquen y compartan datos con otras aplicaciones.
- **Administrador de recursos** : brinda acceso a recursos incrustados sin código, como cadenas, configuraciones de color y diseños de interfaz de usuario.
- **Administrador de notificaciones** : permite que las aplicaciones muestren alertas y notificaciones al usuario.
- **View System** : un conjunto extensible de vistas que se utiliza para crear interfaces de usuario de aplicaciones.

Aplicaciones

Encontrarás todas las aplicaciones de Android en la capa superior. Escribirás tu aplicación para que se instale solo en esta capa. Ejemplos de tales aplicaciones son Contactos, Libros, Navegador, Juegos, etc.

Componentes de la aplicación

Los componentes de la aplicación son los bloques de construcción esenciales de una aplicación de Android. Estos componentes están débilmente acoplados por el archivo de manifiesto de la aplicación *AndroidManifest.xml* que describe cada componente de la aplicación y cómo interactúan.



Hay cuatro componentes principales que se pueden utilizar dentro de una aplicación de Android:

No Señor	Componentes y descripción
1	Activities Dictan la interfaz de usuario y manejan la interacción del usuario con la pantalla del teléfono inteligente.
2	Services Manejan el procesamiento en segundo plano asociado con una aplicación.
3	Broadcast Receivers Manejan la comunicación entre el sistema operativo Android y las aplicaciones.
4	Content Providers Manejan problemas de administración de datos y bases de datos.

Activities

Una actividad representa una sola pantalla con una interfaz de usuario; en resumen, Actividad realiza acciones en la pantalla. Por ejemplo, una aplicación de correo electrónico puede tener una actividad que muestre una lista de correos electrónicos nuevos, otra actividad para redactar un correo electrónico y otra actividad para leer correos electrónicos. Si una aplicación tiene más de una actividad, entonces una de ellas debe marcarse como la actividad que se presenta cuando se inicia la aplicación.

Una actividad se implementa como una subclase de la clase **Actividad** de la siguiente manera:

```
public class MainActivity extends Activity {
}
```

Services

Un servicio es un componente que se ejecuta en segundo plano para realizar operaciones de larga duración. Por ejemplo, un servicio puede reproducir música en segundo plano mientras el usuario está en una aplicación diferente, o puede obtener datos a través de la red sin bloquear la interacción del usuario con una actividad.

Un servicio se implementa como una subclase de la clase de **servicio** de la siguiente manera:

```
public class MyService extends Service {  
}
```

Broadcast Receivers

Los receptores de difusión simplemente responden a los mensajes de difusión de otras aplicaciones o del sistema. Por ejemplo, las aplicaciones también pueden iniciar transmisiones para que otras aplicaciones sepan que se han descargado algunos datos en el dispositivo y están disponibles para su uso, por lo que este es un receptor de transmisión que interceptará esta comunicación e iniciará la acción apropiada.

Un receptor de difusión se implementa como una subclase de la clase **BroadcastReceiver** y cada mensaje se difunde como un objeto **Intent**.

```
public class MyReceiver extends BroadcastReceiver {  
    public void onReceive(context,intent){}  
}
```

Content Providers

Un componente de proveedor de contenido proporciona datos de una aplicación a otras bajo petición. Estas solicitudes son manejadas por los métodos de la clase *ContentResolver*. Los datos pueden almacenarse en el sistema de archivos, la base de datos o en cualquier otro lugar.

Un proveedor de contenido se implementa como una subclase de la clase **ContentProvider** y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones.

```
public class MyContentProvider extends ContentProvider {  
    public void onCreate(){}  
}
```

Repasaremos estas etiquetas en detalle mientras cubrimos los componentes de la aplicación en capítulos individuales.

Componentes adicionales

Hay componentes adicionales que se utilizarán en la construcción de las entidades mencionadas, su lógica y el cableado entre ellas. Estos componentes son:

S.	Componentes y descripción
No	

- 1 **Fragments**
Representa una parte de la interfaz de usuario en una actividad.
- 2 **Views**
Elementos de la interfaz de usuario que se dibujan en pantalla, incluidos botones, listas de formularios, etc.
- 3 **Layouts**
Ver jerarquías que controlan el formato de pantalla y la apariencia de las vistas.
- 4 **Intents**
Mensajes de cableado de componentes juntos.
- 5 **Resources**
Elementos externos, como cadenas, constantes e imágenes dibujables.
- 6 **Manifest**
Archivo de configuración de la aplicación.

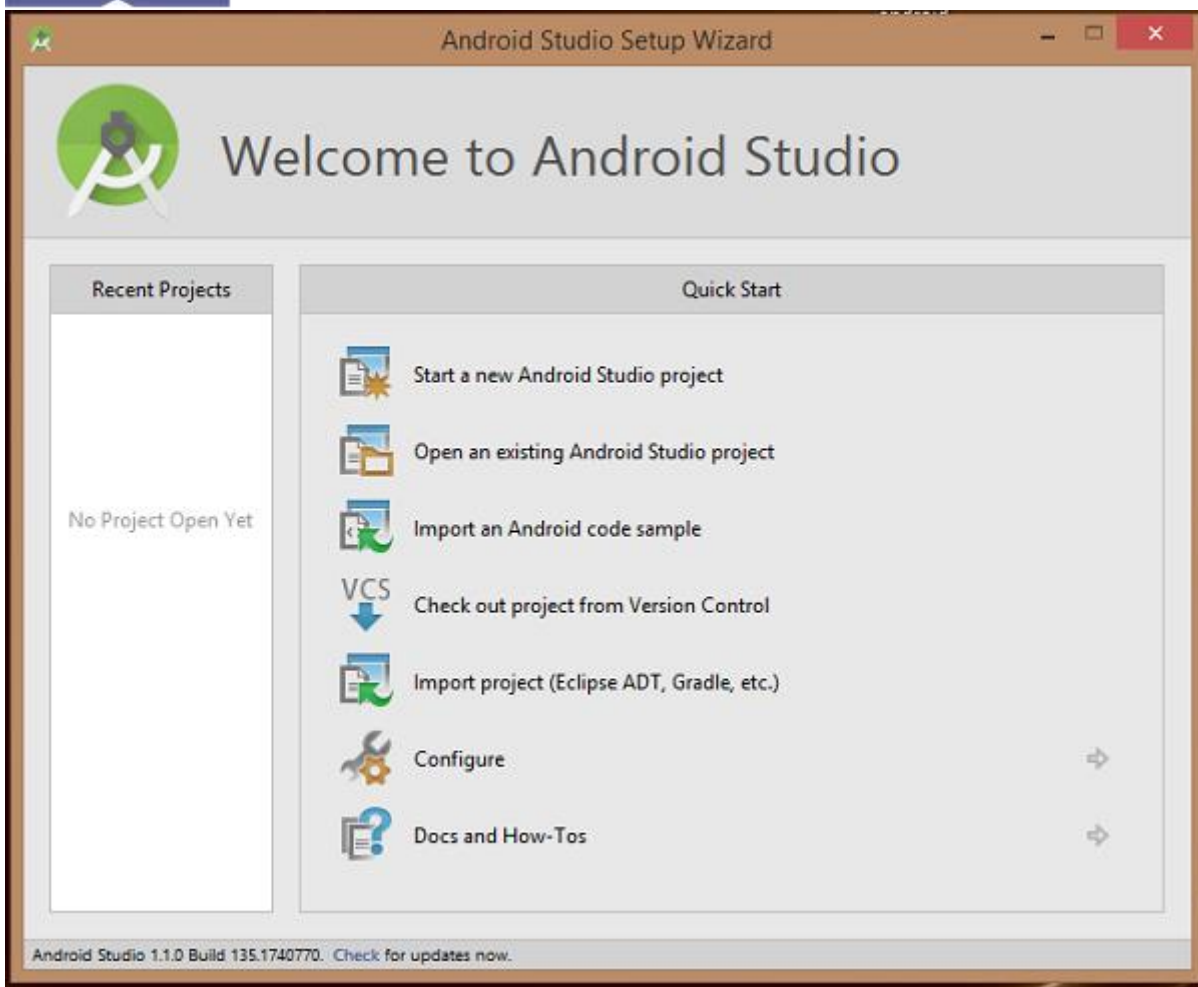
Ejemplo de Hello World

Comencemos la programación real con Android Framework. Antes de comenzar a escribir su primer ejemplo con el SDK de Android, debe asegurarse de haber configurado correctamente su entorno de desarrollo de Android, como se explica en [Android - Tutorial de configuración del entorno](#) . También asumo que tienes un poco de conocimiento práctico con Android Studio.

Así que procedamos a escribir una sencilla aplicación para Android que imprimirá "¡Hola mundo!".

Crear aplicación de Android

El primer paso es crear una aplicación de Android simple usando Android Studio. Cuando haga clic en el icono de estudio de Android, se mostrará la pantalla como se muestra a continuación



Puede iniciar el desarrollo de su aplicación llamando a iniciar un nuevo proyecto de estudio de Android. en un nuevo marco de instalación debe preguntar el nombre de la aplicación, la información del paquete y la ubicación del proyecto.



Create New Project

New Project
Android Studio

Configure your new project

Application name:

Company Domain:

Package name: [Edit](#)

Project location: [...](#)

Please enter an application name (shown in launcher)

[Previous](#) [Next](#) [Cancel](#) [Finish](#)

Después de ingresar el nombre de la aplicación, se llamará seleccione los factores de forma en los que se ejecuta su aplicación, aquí debe especificar el SDK mínimo, en nuestro tutorial, lo he declarado como API23: Android 6.0 (Marshmallow) -

Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet
Minimum SDK: [Help me choose](#)
Lower API levels target more devices, but have fewer features available.
By targeting API 23 and later, your app will run on approximately 4.7% of the devices that are active on the Google Play Store.

☐ Wear
Minimum SDK:

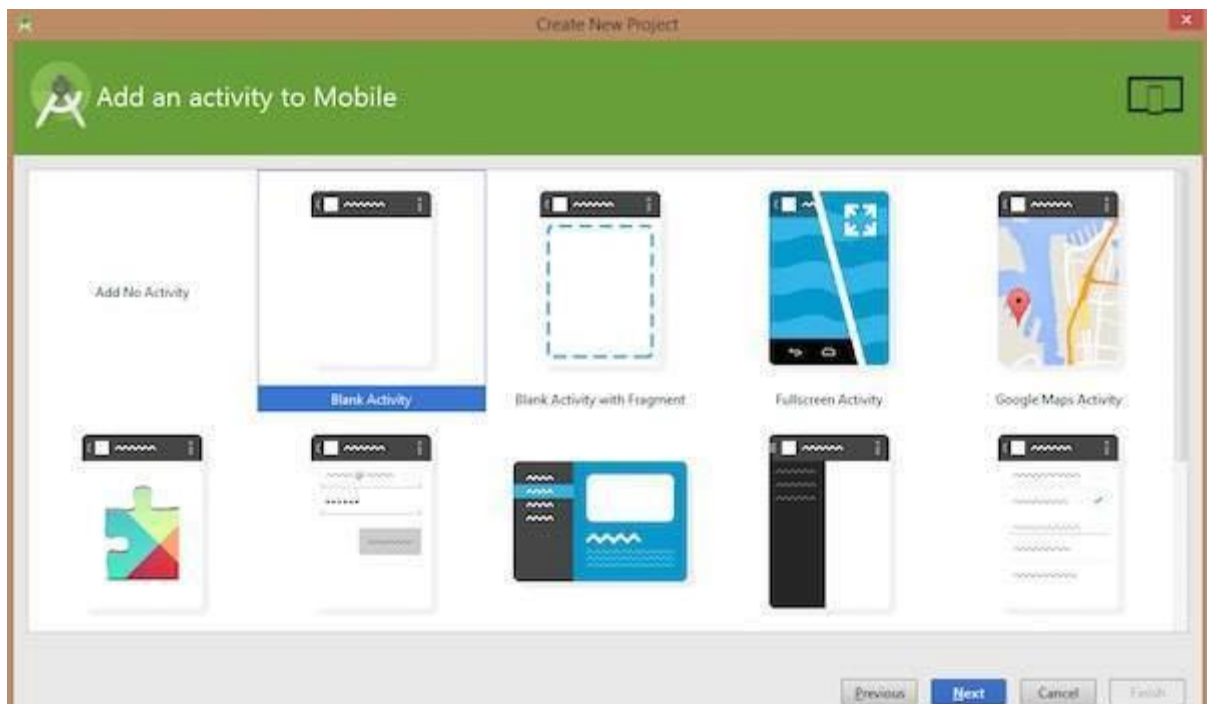
☐ TV
Minimum SDK:

☐ Android Auto

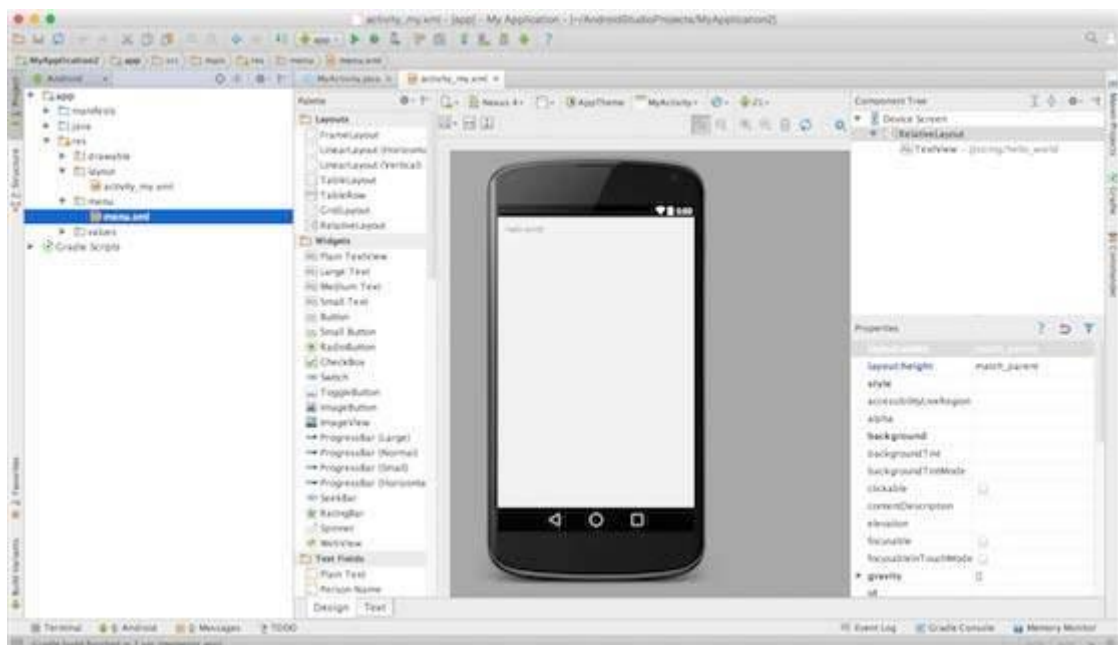
☐ Glass
Minimum SDK:

[Previous](#) [Next](#) [Cancel](#) [Finish](#)

El siguiente nivel de instalación debe contener la selección de la actividad para el móvil, especifica el diseño predeterminado para las Aplicaciones.

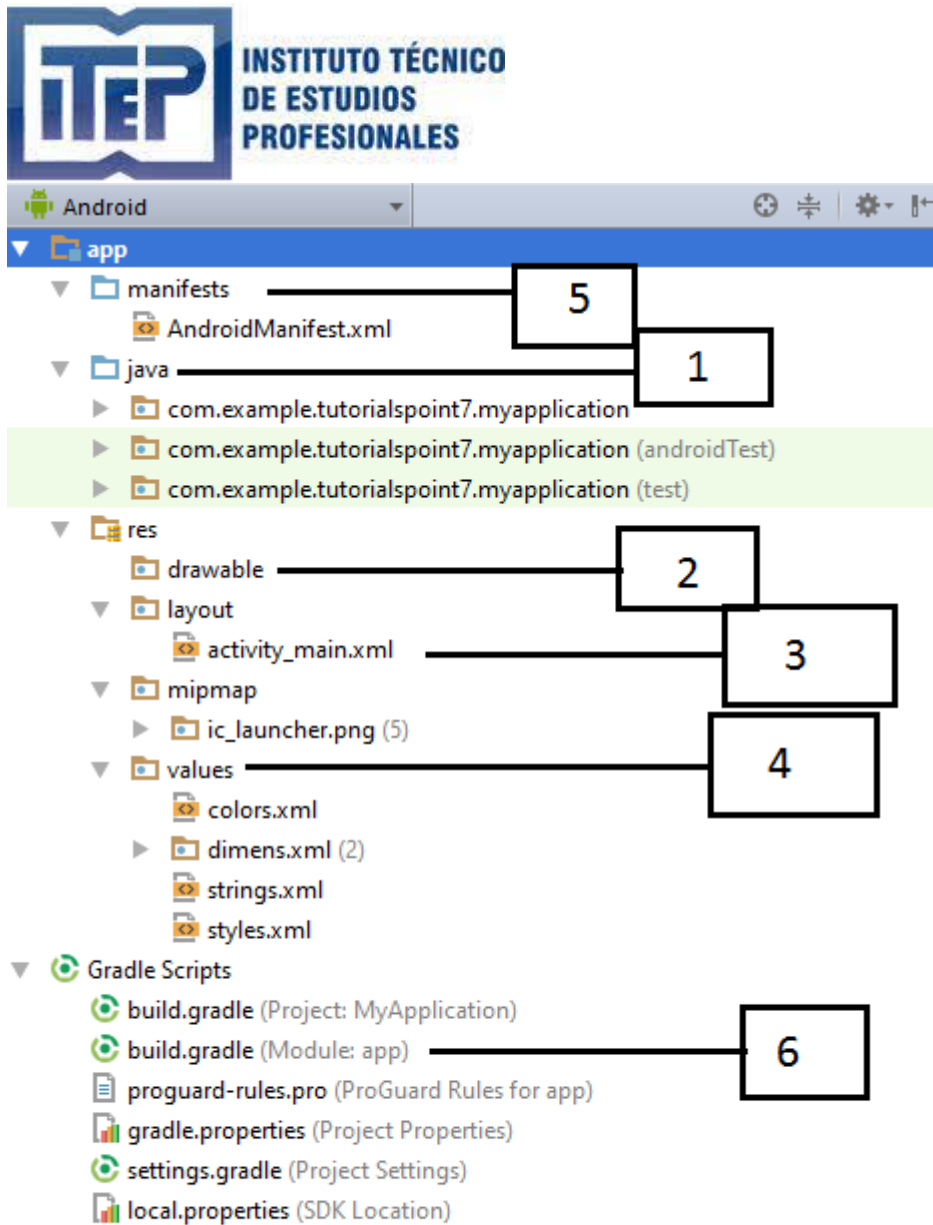


En la etapa final, será una herramienta de desarrollo abierta para escribir el código de la aplicación.



Anatomía de la aplicación de Android

Antes de ejecutar su aplicación, debe conocer algunos directorios y archivos en el proyecto de Android:



No
Señor.

Carpeta, archivo y descripción

1

Java

Este contiene los archivos fuente **.java** para su proyecto. De forma predeterminada, incluye un archivo fuente *MainActivity.java* que tiene una clase de actividad que se ejecuta cuando se inicia la aplicación mediante el icono de la aplicación.

2

res / drawable-hdpi

Este es un directorio para objetos dibujables que están diseñados para pantallas de alta densidad.

3

res / layout

Este es un directorio de archivos que definen la interfaz de usuario de su aplicación.

4 **res / values**

Este es un directorio para otros archivos XML que contienen una colección de recursos, como cadenas y definiciones de colores.

5 **AndroidManifest.xml**

Este es el archivo de manifiesto que describe las características fundamentales de la aplicación y define cada uno de sus componentes.

6 **Build.gradle**

Este es un archivo generado automáticamente que contiene `compileSdkVersion`, `buildToolsVersion`, `applicationId`, `minSdkVersion`, `targetSdkVersion`, `versionCode` y `versionName`

La siguiente sección le dará una breve descripción de los archivos importantes de la aplicación.

El archivo de actividad principal

El código de actividad principal es un archivo Java **MainActivity.java** . Este es el archivo de aplicación real que finalmente se convierte en un ejecutable de Dalvik y ejecuta su aplicación. A continuación se muestra el código predeterminado generado por el asistente de aplicaciones para *Hello World!* aplicación -

```
package com.example.helloworld;

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);
    }
}
```

Aquí, *R.layout.activity_main* se refiere al archivo *activity_main.xml* ubicado en la carpeta *res / layout* . El método *onCreate ()* es uno de los muchos métodos que se calculan cuando se carga una actividad.

El archivo manifest

Cualquiera que sea el componente que desarrolle como parte de su aplicación, debe declarar todos sus componentes en un *manifest.xml* que reside en la raíz del directorio del proyecto de la aplicación. Este archivo funciona como una interfaz entre el sistema operativo Android y su aplicación, por lo que si no declara su componente en este archivo, el sistema operativo no lo considerará. Por ejemplo, un archivo de manifiesto predeterminado se verá como el siguiente archivo:

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

    <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>
```

Aquí, las etiquetas `<application> ... </application>` incluían los componentes relacionados con la aplicación. Atributo *android: el icono* apuntará al icono de la aplicación disponible en *res / drawable-hdpi*. La aplicación usa la imagen llamada *ic_launcher.png* ubicada en las carpetas dibujables

La etiqueta `<activity>` se usa para especificar una actividad y el atributo *android: name* especifica el nombre de la clase completamente calificado de la subclase *Activity* y los atributos *android: label* especifican una cadena para usar como etiqueta para la actividad. Puede especificar varias actividades utilizando etiquetas `<activity>`.

La **acción** para el filtro de intención se llama *android.intent.action.MAIN* para indicar que esta actividad sirve como punto de entrada para la aplicación. La **categoría** para el filtro de intención se llama *android.intent.category.LAUNCHER* para indicar que la aplicación se puede iniciar desde el icono de inicio del dispositivo.

El *@string* se refiere a la *strings.xml* archivo explica a continuación. Por lo tanto, *@ string / app_name* se refiere a la cadena de *app_name* definida en el

archivo `strings.xml`, que es "HelloWorld". De manera similar, otras cadenas se completan en la aplicación.

A continuación se muestra la lista de etiquetas que usará en su archivo de manifiesto para especificar diferentes componentes de la aplicación de Android:

- `<activity>` elementos para actividades
- `<service>` elementos para servicios
- `<receiver>` elementos para receptores de difusión
- `<provider>` elementos para proveedores de contenido

El archivo strings

El archivo **strings.xml** se encuentra en la carpeta `res / values` y contiene todo el texto que usa su aplicación. Por ejemplo, los nombres de botones, etiquetas, texto predeterminado y tipos similares de cadenas se incluyen en este archivo. Este archivo es responsable de su contenido textual. Por ejemplo, un archivo de cadenas predeterminado se verá como el siguiente archivo:

```
<resources>
  <string name="app_name">HelloWorld</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="title_activity_main">MainActivity</string>
</resources>
```

El archivo layout

El **activity_main.xml** es un archivo de diseño disponible en `res / layout` directorio, que hace referencia a su aplicación en la construcción de su interfaz. Modificará este archivo con mucha frecuencia para cambiar el diseño de su aplicación. Para su "¡Hola mundo!" aplicación, este archivo tendrá el siguiente contenido relacionado con el diseño predeterminado:

```
<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" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />
```



```
</RelativeLayout>
```

Este es un ejemplo de *RelativeLayout* simple que estudiaremos en un capítulo aparte. El *TextView* es un control de Android que se usa para construir la GUI y tiene varios atributos como *android: layout_width* , *android: layout_height*, etc. que se utilizan para establecer su ancho y alto, etc. El *@string* se refiere al archivo *strings.xml* ubicado en la carpeta *res / valores*. Por lo tanto, *@ string / hello_world* se refiere a la cadena de saludo definida en el archivo *strings.xml*, que es "¡Hola mundo!".

Ejecutando la Aplicación

¡Intentemos ejecutar nuestro **Hello World!** aplicación que acabamos de crear. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:



¡¡¡Felicidades!!! ha desarrollado su primera aplicación de Android y ahora solo sigue el resto del tutorial paso a paso para convertirse en un gran desarrollador de Android. Todo lo mejor.

Organización y acceso a los recursos de Android

Hay muchos más elementos que puede utilizar para crear una buena aplicación de Android. Además de la codificación para la aplicación, se ocupa de varios otros **recursos**, como el contenido estático que utiliza su código, como mapas de bits, colores, definiciones de diseño, cadenas de interfaz de usuario, instrucciones de animación y más. Estos recursos siempre se mantienen por separado en varios subdirectorios en el directorio **res /** del proyecto.

Este tutorial le explicará cómo puede organizar los recursos de su aplicación, especificar recursos alternativos y acceder a ellos en sus aplicaciones.

Organizar recursos en Android Studio

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
    java/
      MainActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

No
Señor.

Directorio y tipo de recurso

- 1 **anim /**
Archivos XML que definen animaciones de propiedades. Se guardan en la carpeta **res / anim /** y se accede a ellos desde la clase **R.anim** .
- 2 **color/**
Archivos XML que definen una lista de colores de estado. Se guardan en **res / color /** y se accede desde la clase **R.color** .
- 3 **drawable /**

Archivos de imagen como .png, .jpg, .gif o archivos XML que se compilan en mapas de bits, listas de estado, formas, animación dibujable. Se guardan en `res / drawable /` y se accede a ellos desde la clase **R.drawable** .

4

layout/

Archivos XML que definen un diseño de interfaz de usuario. Se guardan en `res / layout /` y se accede a ellos desde la clase **R.layout** .

5

menu/

Archivos XML que definen los menús de la aplicación, como un menú de opciones, un menú contextual o un submenú. Se guardan en `res / menu /` y se accede a ellos desde la clase **R.menu** .

6

raw/

Archivos arbitrarios para guardar en su forma original. *Debe llamar a `Resources.openRawResource ()` con el ID de recurso, que es `R.raw.filename` para abrir dichos archivos sin procesar.*

7

values/

Archivos XML que contienen valores simples, como cadenas, números enteros y colores. Por ejemplo, aquí hay algunas convenciones de nombres de archivos para los recursos que puede crear en este directorio:

- `arrays.xml` para matrices de recursos y se accede desde la clase **R.array** .
- `integers.xml` para enteros de recursos y se accede desde la clase **R.integer** .
- `bools.xml` para el booleano de recursos y se accede desde la clase **R.bool** .
- `colors.xml` para los valores de color, y se accede desde la clase **R.color** .
- `dimens.xml` para valores de dimensión y se accede desde la clase **R.dimen** .
- `strings.xml` para valores de cadena y se accede desde la clase **R.string** .
- `styles.xml` para estilos, y se accede desde la clase **R.style** .

8

xml /

Archivos XML arbitrarios que se pueden leer en tiempo de ejecución llamando a `Resources.getXML ()` . Aquí puede guardar varios archivos de configuración que se utilizarán en tiempo de ejecución.

Recursos alternativos

Su aplicación debe proporcionar recursos alternativos para admitir configuraciones de dispositivo específicas. Por ejemplo, debe incluir recursos dibujables alternativos (es decir, imágenes) para diferentes resoluciones de pantalla y recursos de cadenas alternativos para diferentes idiomas. En tiempo de ejecución, Android detecta la configuración actual del dispositivo y carga los recursos adecuados para su aplicación.

Para especificar alternativas específicas de configuración para un conjunto de recursos, siga los siguientes pasos:

- Cree un nuevo directorio en `res /` con el formato **<resources_name> - <config_qualifier>**. Aquí **resources_name** será cualquiera de los recursos mencionados en la tabla anterior, como diseño, dibujable, etc. El **qualifier** especificará una configuración individual para la que se utilizarán estos recursos. Puede consultar la documentación oficial para obtener una lista completa de los calificadores para diferentes tipos de recursos.
- Guarde los recursos alternativos respectivos en este nuevo directorio. Los archivos de recursos deben tener el mismo nombre que los archivos de recursos predeterminados como se muestra en el siguiente ejemplo, pero estos archivos tendrán contenido específico para la alternativa. Por ejemplo, aunque el nombre del archivo de imagen será el mismo pero para una pantalla de alta resolución, su resolución será alta.

A continuación se muestra un ejemplo que especifica imágenes para una pantalla predeterminada e imágenes alternativas para una pantalla de alta resolución.

```
MyProject/  
  app/  
    manifest/  
      AndroidManifest.xml  
  java/  
    MainActivity.java  
  res/  
    drawable/  
      icon.png  
      background.png  
    drawable-hdpi/  
      icon.png  
      background.png  
    layout/  
      activity_main.xml  
      info.xml  
    values/  
      strings.xml
```

A continuación se muestra otro ejemplo que especifica el diseño para un idioma predeterminado y un diseño alternativo para el idioma específico.

```
MyProject/  
  app/  
    manifest/  
      AndroidManifest.xml  
  java/  
    MainActivity.java  
  res/  
    drawable/  
      icon.png  
      background.png  
    drawable-hdpi/  
      icon.png  
      background.png  
    layout/  
      activity_main.xml  
      info.xml  
    layout-ar/  
      main.xml  
    values/  
      strings.xml
```

Acceso a recursos

Durante el desarrollo de su aplicación, deberá acceder a los recursos definidos en su código o en sus archivos XML de diseño. La siguiente sección explica cómo acceder a sus recursos en ambos escenarios:

Acceso a recursos en código

Cuando se compila su aplicación de Android, se genera una clase **R**, que contiene ID de recursos para todos los recursos disponibles en su directorio **res** /. Puede usar la clase R para acceder a ese recurso usando el subdirectorio y el nombre del recurso o directamente el ID del recurso.

Ejemplo

Para acceder a *res / drawable / myimage.png* y configurar un `ImageView`, usará el siguiente código:

```
ImageView imageView = (ImageView)  
findViewById(R.id.myimageview);  
imageView.setImageResource(R.drawable.myimage);
```

Aquí la primera línea del código hace uso de *R.id.myimageview* para obtener `ImageView` definido con id *myimageview* en un archivo de diseño. La segunda línea de código hace uso de *R.drawable.myimage* para obtener una imagen con el nombre **myimage** disponible en el subdirectorio **drawable** en **/ res** .

Ejemplo

Considere el siguiente ejemplo donde *res / values / strings.xml* tiene la siguiente definición:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, World!</string>
</resources>
```

Ahora puede configurar el texto en un objeto `TextView` con ID `msg` usando una ID de recurso de la siguiente manera:

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Ejemplo

Considere un diseño *res / layout / activity_main.xml* con la siguiente definición:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />

</LinearLayout>
```

Este código de aplicación cargará este diseño para una actividad, en el método `onCreate ()` de la siguiente manera:

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

Acceso a recursos en XML



Considere el siguiente archivo XML de recursos *res / values / strings.xml* que incluye un recurso de color y un recurso de cadena:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="opaque_red">#f00</color>
    <string name="hello">Hello!</string>
</resources>
```

Ahora puede usar estos recursos en el siguiente archivo de diseño para establecer el color del texto y la cadena de texto de la siguiente manera:

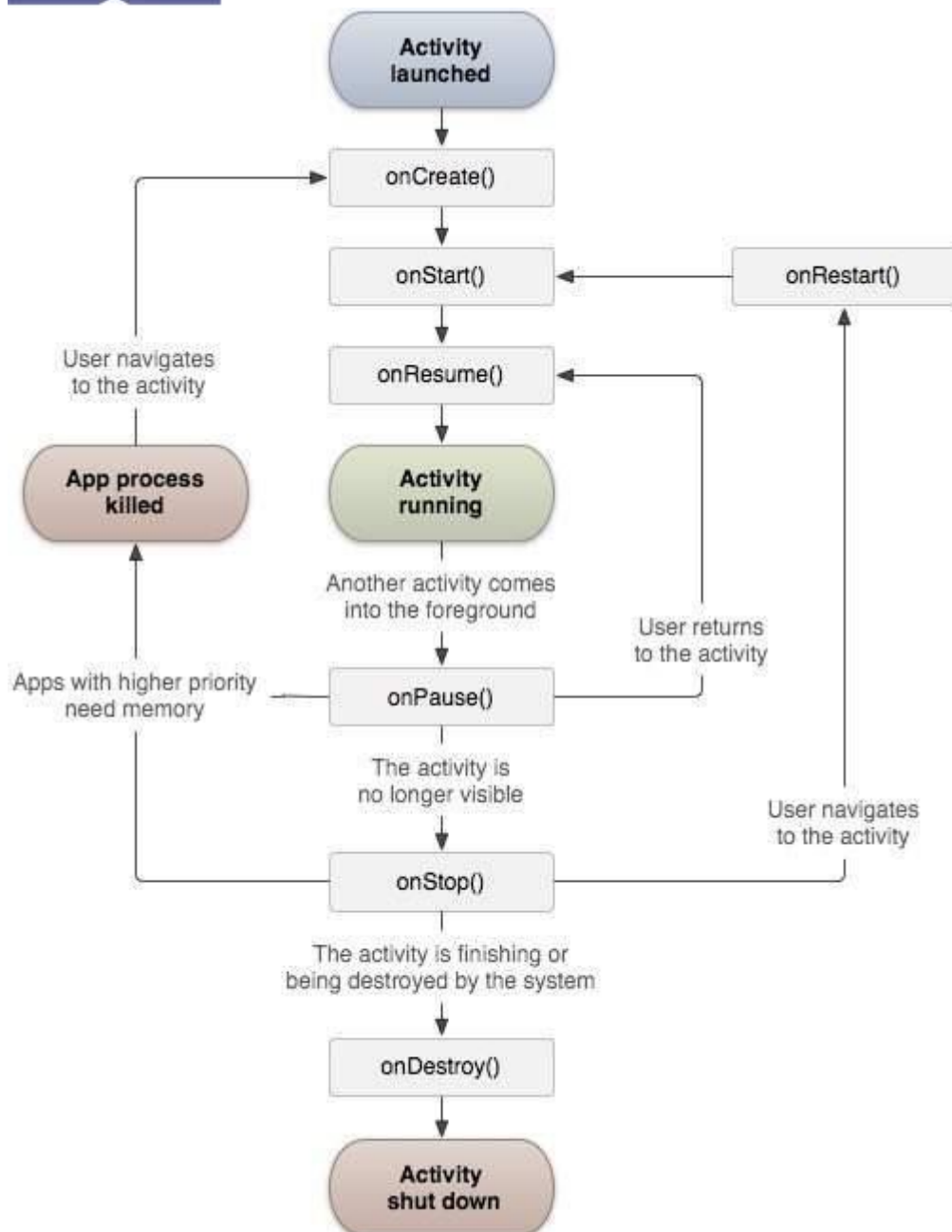
```
<?xml version="1.0" encoding="utf-8"?>
<EditText
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Ahora, si vuelve a pasar por el capítulo anterior, donde he explicado **¡Hola mundo!** ejemplo, y estoy seguro de que comprenderá mejor todos los conceptos explicados en este capítulo. Así que recomiendo encarecidamente consultar el capítulo anterior para ver un ejemplo de trabajo y comprobar cómo he utilizado varios recursos a un nivel muy básico.

Actividades

Una actividad representa una sola pantalla con una interfaz de usuario como una ventana o un marco de Java. La actividad de Android es la subclase de la clase `ContextThemeWrapper`.

Si ha trabajado con el lenguaje de programación C, C++ o Java, entonces debe haber visto que su programa comienza desde la función **main ()**. De manera muy similar, el sistema Android inicia su programa con una **actividad** que comienza con una llamada al método de devolución de llamada *onCreate ()*. Hay una secuencia de métodos de devolución de llamada que inician una actividad y una secuencia de métodos de devolución de llamada que destruyen una actividad como se muestra en el siguiente diagrama de ciclo de vida de la actividad: (*imagen cortesía: android.com*)



La clase Activity define las siguientes devoluciones de llamada, es decir, eventos. No es necesario implementar todos los métodos de devolución de llamada. Sin embargo, es importante que comprenda cada uno e implemente aquellos que aseguren que su aplicación se comporte de la manera que esperan los usuarios.

No
Señor

Devolución de llamada y descripción

1

onCreate ()

Esta es la primera devolución de llamada y se llama cuando se crea la actividad por primera vez.

2 **onStart ()**

Esta devolución de llamada se llama cuando la actividad se vuelve visible para el usuario.

3 **onResume()**

Se llama cuando el usuario comienza a interactuar con la aplicación.

4 **onPause ()**

La actividad pausada no recibe la entrada del usuario y no puede ejecutar ningún código y se llama cuando la actividad actual se está pausando y se reanuda la actividad anterior.

5 **onStop ()**

Esta devolución de llamada se llama cuando la actividad ya no es visible.

6 **onDestroy ()**

Esta devolución de llamada se llama antes de que el sistema destruya la actividad.

7 **onRestart ()**

Esta devolución de llamada se llama cuando la actividad se reinicia después de detenerla.

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar el ciclo de vida de la actividad de la aplicación de Android. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla como <i>HelloWorld</i> en un paquete <i>com.example.helloworld</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el archivo de actividad principal <i>MainActivity.java</i> como se explica a continuación. Mantenga el resto de los archivos sin cambios.

- 3 Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.helloworld / MainActivity.java** . Este archivo incluye cada uno de los métodos fundamentales del ciclo de vida. El método **Log.d ()** se ha utilizado para generar mensajes de registro:

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
    String msg = "Android : ";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }

    /** Called when the activity is about to become visible.
    */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "The onStart() event");
    }

    /** Called when the activity has become visible. */
    @Override
    protected void onResume() {
        super.onResume();
        Log.d(msg, "The onResume() event");
    }

    /** Called when another activity is taking focus. */
    @Override
    protected void onPause() {
        super.onPause();
        Log.d(msg, "The onPause() event");
    }

    /** Called when the activity is no longer visible. */
    @Override
    protected void onStop() {
```

```
        super.onStop();
        Log.d(msg, "The onStop() event");
    }

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(msg, "The onDestroy() event");
    }
}
```

Una clase de actividad carga todo el componente de la interfaz de usuario utilizando el archivo XML disponible en la carpeta *res / layout* del proyecto. La siguiente declaración carga los componentes de la interfaz de usuario del archivo *res / layout / activity_main.xml* :

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

Una aplicación puede tener una o más actividades sin restricciones. Cada actividad que defina para su aplicación debe declararse en su archivo *AndroidManifest.xml* y la actividad principal de su aplicación debe declararse en el manifiesto con un `<intent-filter>` que incluye la acción MAIN y la categoría LAUNCHER de la siguiente manera:

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

    <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>
```

Si la acción MAIN o la categoría LAUNCHER no se declaran para una de sus actividades, el icono de su aplicación no aparecerá en la lista de aplicaciones de la pantalla de inicio.

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar de la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, mostrará la ventana Emulador y debería ver los siguientes mensajes de registro en la ventana **LogCat** en Android Studio:

```
08-23 10:32:07.682 4480-4480/com.example.helloworld
D/Android :: The onCreate() event
08-23 10:32:07.683 4480-4480/com.example.helloworld
D/Android :: The onStart() event
08-23 10:32:07.685 4480-4480/com.example.helloworld
D/Android :: The onResume() event
```




Intentemos hacer clic en el botón de bloqueo de pantalla en el emulador de Android y generará los siguientes mensajes de eventos en la ventana de **LogCat** en Android Studio:

```
08-23 10:32:53.230 4480-4480/com.example.helloworld
D/Android :: The onPause() event
08-23 10:32:53.294 4480-4480/com.example.helloworld
D/Android :: The onStop() event
```

Intentemos nuevamente desbloquear su pantalla en el emulador de Android y generará los siguientes mensajes de eventos en la ventana de **LogCat** en el estudio de Android:

```
08-23 10:34:41.390 4480-4480/com.example.helloworld
D/Android :: The onStart() event
08-23 10:34:41.392 4480-4480/com.example.helloworld
D/Android :: The onResume() event
```

A continuación, intentemos nuevamente hacer clic  en el botón Atrás en el emulador de Android y generará los siguientes mensajes de eventos en la ventana de **LogCat** en el estudio de Android y esto completa el ciclo de vida de la actividad para una aplicación de Android.

```
08-23 10:37:24.806 4480-4480/com.example.helloworld
D/Android :: The onPause() event
08-23 10:37:25.668 4480-4480/com.example.helloworld
D/Android :: The onStop() event
08-23 10:37:25.669 4480-4480/com.example.helloworld
D/Android :: The onDestroy() event
```

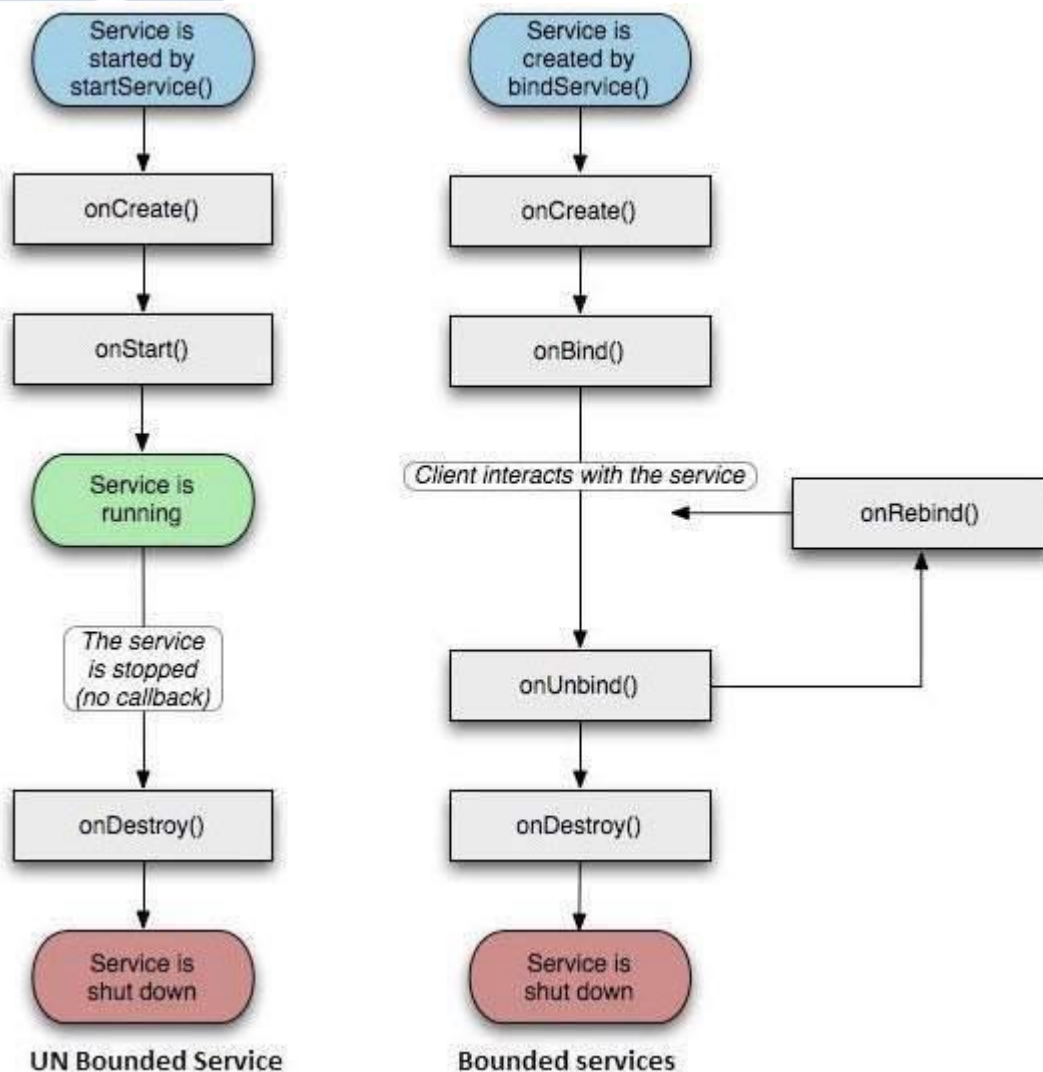
Servicios

Un **service** es un componente que se ejecuta en segundo plano para realizar operaciones de larga duración sin necesidad de interactuar con el usuario y

funciona incluso si se destruye la aplicación. Un servicio puede tener esencialmente dos estados:

No Señor.	Estado y descripción
1	<p>Started</p> <p>Un servicio se inicia cuando un componente de la aplicación, como una actividad, lo inicia llamando a <code>startService ()</code> . Una vez iniciado, un servicio puede ejecutarse en segundo plano indefinidamente, incluso si el componente que lo inició se destruye.</p>
2	<p>Bound</p> <p>Un servicio está bound cuando un componente de la aplicación se une a él llamando a <code>bindService ()</code> . Un servicio vinculado ofrece una interfaz cliente-servidor que permite que los componentes interactúen con el servicio, envíen solicitudes, obtengan resultados e incluso lo hagan a través de procesos con comunicación entre procesos (IPC).</p>

Un servicio tiene métodos de devolución de llamada de ciclo de vida que puede implementar para monitorear los cambios en el estado del servicio y puede realizar el trabajo en la etapa apropiada. El siguiente diagrama de la izquierda muestra el ciclo de vida cuando el servicio se crea con `startService ()` y el diagrama de la derecha muestra el ciclo de vida cuando el servicio se crea con `bindService ()`: (*imagen cortesía: android.com*)



Para crear un servicio, cree una clase Java que amplíe la clase base de Servicio o una de sus subclases existentes. La clase base **Service** define varios métodos de devolución de llamada y los más importantes se dan a continuación. No es necesario implementar todos los métodos de devolución de llamada. Sin embargo, es importante que comprenda cada uno e implemente aquellos que garanticen que su aplicación se comporte de la manera que esperan los usuarios.

No
Señor.

Devolución de llamada y descripción

1

onStartCommand ()

El sistema llama a este método cuando otro componente, como una actividad, solicita que se inicie el servicio, llamando a `startService ()`. Si implementa este método, es su responsabilidad detener el servicio cuando *termine* su trabajo, llamando a los métodos `stopSelf ()` o `stopService ()`.

2

onBind ()

El sistema llama a este método cuando otro componente quiere vincularse con el servicio llamando a *bindService ()* . Si implementa este método, debe proporcionar una interfaz que los clientes usen para comunicarse con el servicio, devolviendo un objeto *IBinder* . Siempre debe implementar este método, pero si no desea permitir el enlace, debe devolver un *valor nulo* .

3

onUnbind ()

El sistema llama a este método cuando todos los clientes se han desconectado de una interfaz en particular publicada por el servicio.

4

onRebind ()

El sistema llama a este método cuando nuevos clientes se han conectado al servicio, después de haber sido notificado previamente que todos se habían desconectado en su *onUnbind (Intent)* .

5

onCreate ()

El sistema llama a este método cuando el servicio se crea por primera vez utilizando *onStartCommand ()* o *onBind ()* . Esta llamada es necesaria para realizar una configuración única.

6

onDestroy ()

El sistema llama a este método cuando el servicio ya no se utiliza y se está destruyendo. Su servicio debe implementar esto para limpiar cualquier recurso como hilos, oyentes registrados, receptores, etc.

El siguiente servicio de esqueleto demuestra cada uno de los métodos del ciclo de vida:

```
package com.tutorial;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

    /** indicates how to behave if the service is killed */
    int mStartMode;

    /** interface for clients that bind */
    IBinder mBinder;

    /** indicates whether onRebind should be used */
    boolean mAllowRebind;
```



```
/** Called when the service is being created. */
@Override
public void onCreate() {

}

/** The service is starting, due to a call to
startService() */
@Override
public int onStartCommand(Intent intent, int flags, int
startId) {
    return mStartMode;
}

/** A client is binding to the service with bindService()
*/
@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

/** Called when all clients have unbound with
unbindService() */
@Override
public boolean onUnbind(Intent intent) {
    return mAllowRebind;
}

/** Called when a client is binding to the service with
bindService() */
@Override
public void onRebind(Intent intent) {

}

/** Called when The service is no longer used and is being
destroyed */
@Override
public void onDestroy() {

}
}
```

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propio servicio de Android. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>Mi aplicación</i> en un paquete <i>com.example.myapplication</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el archivo de actividad principal <i>MainActivity.java</i> para agregar los métodos <i>startService ()</i> y <i>stopService ()</i> .
3	Cree un nuevo archivo java <i>MyService.java</i> bajo el paquete <i>com.example.My Application</i> . Este archivo tendrá la implementación de métodos relacionados con el servicio de Android.
4	Defina su servicio en el archivo <i>AndroidManifest.xml</i> usando la etiqueta <code><service ... /></code> . Una aplicación puede tener uno o más servicios sin ninguna restricción.
5	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir dos botones en diseño lineal.
6	No es necesario cambiar ninguna constante en el archivo <i>res / values / strings.xml</i> . Android Studio se encarga de los valores de cadena
7	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **MainActivity.java**. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida. Hemos agregado los métodos *startService ()* y *stopService ()* para iniciar y detener el servicio.

```
package com.example.myapplication;

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

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {
    String msg = "Android : ";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.d(msg, "The onCreate() event");
}

public void startService(View view) {
    startService(new Intent(getApplicationContext(),
MyService.class));
}

// Method to stop the service
public void stopService(View view) {
    stopService(new Intent(getApplicationContext(),
MyService.class));
}
}
```

A continuación se muestra el contenido de **MyService.java**. Este archivo puede tener implementación de uno o más métodos asociados con el Servicio según los requisitos. Por ahora vamos a implementar solo dos métodos *onStartCommand()* y *onDestroy()* -

```
package com.example.myapplication;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;

public class MyService extends Service {
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int
startId) {
        // Let it continue running until it is stopped.
        Toast.makeText(this, "Service Started",
Toast.LENGTH_LONG).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

```
        Toast.makeText(this, "Service Destroyed",
        Toast.LENGTH_LONG).show();
    }
}
```

A continuación se muestra el contenido modificado del archivo *AndroidManifest.xml*. Aquí hemos agregado la etiqueta `<service ... />` para incluir nuestro servicio -

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

    <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>

        <service android:name=".MyService" />
    </application>

</manifest>
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** para incluir dos botones:

```
<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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Example of services"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />


<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="Start Services"
    android:onClick="startService"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Stop Services"
    android:id="@+id/button"
    android:onClick="stopService"
    android:layout_below="@+id/button2"
    android:layout_alignLeft="@+id/button2"
    android:layout_alignStart="@+id/button2"
    android:layout_alignRight="@+id/button2"
    android:layout_alignEnd="@+id/button2" />

</RelativeLayout>

```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la

configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:

Ahora, para iniciar su servicio, hagamos clic en el botón **Iniciar servicio**, esto iniciará el servicio y, según nuestra programación en el método `onStartCommand()`, aparecerá un mensaje *Servicio iniciado* en la parte inferior del simulador

Para detener el servicio, puede hacer clic en el botón Detener servicio.

Receptores de difusión

Broadcast Receivers simplemente responden a los mensajes de difusión de otras aplicaciones o del propio sistema. En ocasiones, estos mensajes se denominan eventos o intenciones. Por ejemplo, las aplicaciones también pueden iniciar transmisiones para que otras aplicaciones sepan que se han descargado algunos datos en el dispositivo y están disponibles para su uso, por lo que este es un receptor de transmisión que interceptará esta comunicación e iniciará la acción apropiada.

Hay dos pasos importantes siguientes para que `BroadcastReceiver` funcione para los intentos de transmisión del sistema:

- Creación del receptor de transmisión.
- Registro de receptor de transmisión

Hay un paso adicional en caso de que vaya a implementar sus intenciones personalizadas, entonces tendrá que crear y difundir esas intenciones.

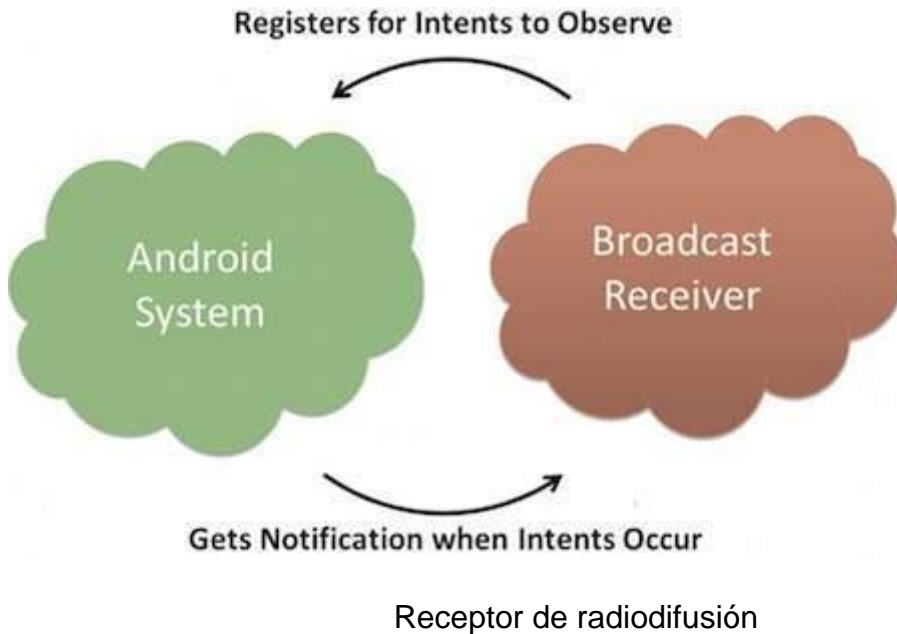
Creación del receptor de transmisión

Un receptor de difusión se implementa como una subclase de la clase **`BroadcastReceiver`** y anula el método `onReceive()` donde cada mensaje se recibe como un parámetro de objeto **`Intent`**.

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(context, "Intent Detected.",  
            Toast.LENGTH_LONG).show();  
    }  
}
```

Registro de receptor de transmisión

Una aplicación escucha las intenciones de transmisión específicas al registrar un receptor de transmisión en el archivo *AndroidManifest.xml*. Considere que vamos a registrar *MyReceiver* para el evento generado por el sistema *ACTION_BOOT_COMPLETED* que es disparado por el sistema una vez que el sistema Android ha completado el proceso de arranque.



```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">

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

    </receiver>
</application>
```

Ahora, cada vez que se *inicie* su dispositivo Android, BroadcastReceiver *MyReceiver* lo interceptará y se ejecutará la lógica implementada dentro de *onReceive()*.

Hay varios eventos generados por el sistema definidos como campos estáticos finales en la clase **Intent**. La siguiente tabla enumera algunos eventos importantes del sistema.

No
Señor

Constante y descripción del evento

- 1 **android.intent.action.BATTERY_CHANGED**
Transmisión fija que contiene el estado de carga, el nivel y otra información sobre la batería.
- 2 **android.intent.action.BATTERY_LOW**
Indica una condición de batería baja en el dispositivo.
- 3 **android.intent.action.BATTERY_OKAY**
Indica que la batería ahora está bien después de estar baja.
- 4 **android.intent.action.BOOT_COMPLETED**
Esto se transmite una vez, después de que el sistema haya terminado de iniciarse.
- 5 **android.intent.action.BUG_REPORT**
Mostrar actividad para informar de un error.
- 6 **android.intent.action.CALL**
Realice una llamada a alguien especificado por los datos.
- 7 **android.intent.action.CALL_BUTTON**
El usuario presionó el botón "llamar" para ir al marcador u otra interfaz de usuario apropiada para realizar una llamada.
- 8 **android.intent.action.DATE_CHANGED**
La fecha ha cambiado.
- 9 **android.intent.action.REBOOT**
Haga que el dispositivo se reinicie.

Difusión de intenciones personalizadas

Si desea que su propia aplicación genere y envíe intents personalizados, tendrá que crear y enviar esos intents utilizando el método `sendBroadcast()` dentro de su clase de actividad. Si usa el método `sendStickyBroadcast(Intent)`, el Intent es **sticky**, lo que significa que el Intent que está enviando permanece después de que se completa la transmisión.

```
public void broadcastIntent(View view) {  
    Intent intent = new Intent();  
    intent.setAction("com.tutorial.CUSTOM_INTENT");  
}
```



```
        sendBroadcast(intent);
    }
}
```

Esta intención *com.tutorial.CUSTOM_INTENT* también se puede registrar de manera similar a como hemos registrado la intención generada por el sistema.

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">

        <intent-filter>
            <action android:name="com.tutorial.CUSTOM_INTENT">
            </action>
        </intent-filter>

    </receiver>
</application>
```

Ejemplo

Este ejemplo le explicará cómo crear *BroadcastReceiver* para interceptar la intención personalizada. Una vez que esté familiarizado con la intención personalizada, puede programar su aplicación para interceptar las intenciones generadas por el sistema. Así que sigamos los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla como <i>Mi aplicación</i> en un paquete <i>com.example.myapplication</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el archivo de actividad principal <i>MainActivity.java</i> para agregar el método <i>broadcastIntent ()</i> .
3	Cree un nuevo archivo java llamado <i>MyReceiver.java</i> en el paquete <i>com.example.tutorialspoint7.myapplication</i> para definir un <i>BroadcastReceiver</i> .
4	Una aplicación puede manejar una o más intenciones personalizadas y del sistema sin restricciones. Cada intento que desee interceptar debe registrarse en su archivo <i>AndroidManifest.xml</i> usando la etiqueta <i><receptor ... /></i>
5	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir un botón para transmitir la intención.

- 6 No es necesario modificar el archivo de cadena, Android Studio se encarga del archivo string.xml.
- 7 Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **MainActivity.java**. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida. Hemos agregado el método *broadcastIntent()* para transmitir una intención personalizada.

```
package com.example.myapplication;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    /** Called when the activity is first created. */
    @Override

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

    // broadcast a custom intent.

    public void broadcastIntent(View view){
        Intent intent = new Intent();
        intent.setAction("com.tutorial.CUSTOM_INTENT");
        sendBroadcast(intent);
    }
}
```

A continuación se muestra el contenido de **MyReceiver.java** :

```
package com.example.myapplication;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

/**
 * Created by TutorialsPoint7 on 8/23/2016.
 */
public class MyReceiver extends BroadcastReceiver{
```

```
@Override
public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Intent Detected.",
        Toast.LENGTH_LONG).show();
}
```

A continuación se muestra el contenido modificado del archivo *AndroidManifest.xml* . Aquí hemos agregado la etiqueta `<receiver ... />` para incluir nuestro servicio:

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

    <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>

        <receiver android:name="MyReceiver">
            <intent-filter>
                <action
                    android:name="com.tutorial.CUSTOM_INTENT">
                </action>
            </intent-filter>

        </receiver>
    </application>

</manifest>
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** para incluir un botón para transmitir nuestra intención personalizada:

```
<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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Example of Broadcast"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials "
        android:textColor="#ff87ff09"
        android:textSize="30dp"
        android:layout_above="@+id/imageButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="40dp" />

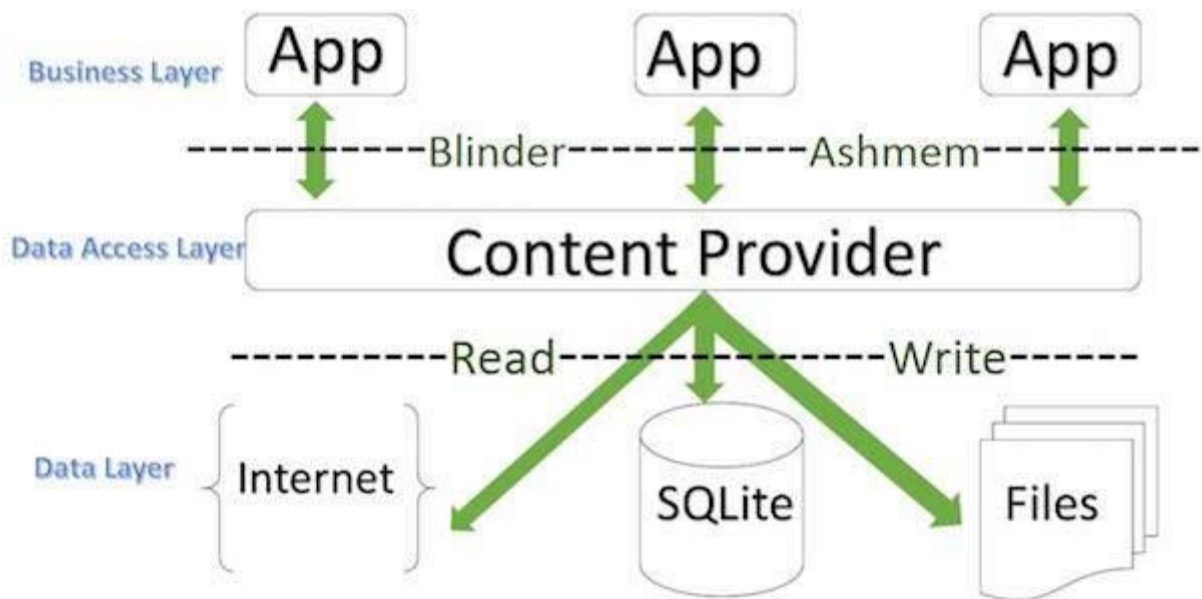
    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:src="@drawable/abc"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:text="Broadcast Intent"
        android:onClick="broadcastIntent"
        android:layout_below="@+id/imageButton"
        android:layout_centerHorizontal="true" />
```

</RelativeLayout>

Proveedores de contenido

Un componente de proveedor de contenido proporciona datos de una aplicación a otras bajo petición. Estas solicitudes son manejadas por los métodos de la clase `ContentResolver`. Un proveedor de contenido puede utilizar diferentes formas de almacenar sus datos y los datos se pueden almacenar en una base de datos, en archivos o incluso en una red.



Proveedor de contenido

A veces es necesario compartir datos entre aplicaciones. Aquí es donde los proveedores de contenido se vuelven muy útiles.

Los proveedores de contenido le permiten centralizar el contenido en un solo lugar y hacer que muchas aplicaciones diferentes accedan a él según sea necesario. Un proveedor de contenido se comporta de manera muy similar a una base de datos donde puede consultarlo, editar su contenido, así como agregar o eliminar contenido usando los métodos `insert()`, `update()`, `delete()` y `query()`. En la mayoría de los casos, estos datos se almacenan en una base de datos **SQLite**.

Un proveedor de contenido se implementa como una subclase de la clase **`ContentProvider`** y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones.

```
public class My Application extends ContentProvider {
}
```



URI de contenido

Para consultar un proveedor de contenido, especifique la cadena de consulta en forma de URI que tiene el siguiente formato:

```
<prefix>://<authority>/<data_type>/<id>
```

Aquí está el detalle de varias partes del URI:

No Señor	Parte Descripción
1	<p>prefix</p> <p>Esto siempre se establece en content: //</p>
2	<p>authority</p> <p>Esto especifica el nombre del proveedor de contenido, por ejemplo, <i>contactos</i>, <i>navegador</i>, etc. Para los proveedores de contenido de terceros, este podría ser el nombre completo, como <i>com.tutorialspoint.statusprovider</i></p>
3	<p>Data_type</p> <p>Esto indica el tipo de datos que proporciona este proveedor en particular. Por ejemplo, si obtiene todos los contactos del proveedor de contenido de <i>Contactos</i>, entonces la ruta de datos sería <i>personas</i> y la URI se vería como este <i>contenido: // contactos / personas</i></p>
4	<p>id</p> <p>Esto especifica el registro específico solicitado. Por ejemplo, si está buscando el número de contacto 5 en el proveedor de contenido de <i>Contactos</i>, entonces el URI se vería así : <i>// contactos / personas / 5</i>.</p>

Crear proveedor de contenido

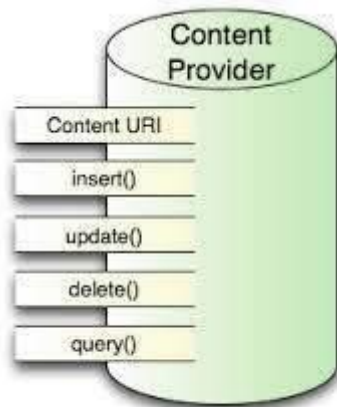
Esto implica una serie de sencillos pasos para crear su propio proveedor de contenido.

- En primer lugar, debe crear una clase de proveedor de contenido que amplíe la clase base *ContentProvider*.
- En segundo lugar, debe definir la dirección URI de su proveedor de contenido que se utilizará para acceder al contenido.
- A continuación, deberá crear su propia base de datos para mantener el contenido. Por lo general, Android usa la base de datos SQLite y el marco necesita anular el método *onCreate ()* que usará el método SQLite Open Helper para crear o abrir la base de datos del proveedor. Cuando se inicia

su aplicación, el controlador *onCreate ()* de cada uno de sus proveedores de contenido se llama en el hilo principal de la aplicación.

- A continuación, deberá implementar consultas del proveedor de contenido para realizar diferentes operaciones específicas de la base de datos.
- Finalmente, registre su proveedor de contenido en su archivo de actividad usando la etiqueta <provider>.

Aquí está la lista de métodos que debe anular en la clase de proveedor de contenido para que su proveedor de contenido funcione:



Proveedor de contenido

- **onCreate ()** Este método se llama cuando se inicia el proveedor.
- **query ()** Este método recibe una solicitud de un cliente. El resultado se devuelve como un objeto Cursor.
- **insert ()** Este método inserta un nuevo registro en el proveedor de contenido.
- **delete ()** Este método elimina un registro existente del proveedor de contenido.
- **update ()** Este método actualiza un registro existente del proveedor de contenido.
- **getType ()** Este método devuelve el tipo MIME de los datos en el URI dado.

Ejemplo

Este ejemplo le explicará cómo crear su propio *ContentProvider*. Así que sigamos los siguientes pasos de manera similar a lo que seguimos al crear el *ejemplo de Hello World*:

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>Mi aplicación</i> en un paquete <i>com.example.MyApplication</i> , con Activity en blanco.
2	Modifique el archivo de actividad principal <i>MainActivity.java</i> para agregar dos nuevos métodos <i>onClickAddName ()</i> y <i>onClickRetrieveStudents ()</i> .
3	Cree un nuevo archivo Java llamado <i>StudentsProvider.java</i> bajo el paquete <i>com.example.MyApplication</i> para definir su proveedor real y los métodos asociados.
4	Registre su proveedor de contenido en su archivo <i>AndroidManifest.xml</i> usando la etiqueta <i><proveedor ... /></i>
5	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir una pequeña GUI para agregar registros de estudiantes.
6	No es necesario cambiar <i>string.xml</i> . Android Studio se encarga del archivo <i>string.xml</i> .
7	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.MyApplication / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida. Hemos agregado dos nuevos métodos *onClickAddName ()* y *onClickRetrieveStudents ()* para manejar la interacción del usuario con la aplicación.

```
package com.example.MyApplication;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;

import android.content.ContentValues;
import android.content.CursorLoader;

import android.database.Cursor;

import android.view.Menu;
import android.view.View;

import android.widget.EditText;
```

```
import android.widget.Toast;

public class MainActivity extends Activity {

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

    public void onClickAddName(View view) {
        // Add a new student record
        ContentValues values = new ContentValues();
        values.put(StudentsProvider.NAME,

((EditText) findViewById(R.id.editText2)).getText().toString()
);

        values.put(StudentsProvider.GRADE,

((EditText) findViewById(R.id.editText3)).getText().toString()
);

        Uri uri = getContentResolver().insert(
            StudentsProvider.CONTENT_URI, values);

        Toast.makeText(getBaseContext(),
            uri.toString(), Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveStudents(View view) {
        // Retrieve student records
        String URL =
"content://com.example.MyApplication.StudentsProvider";

        Uri students = Uri.parse(URL);
        Cursor c = managedQuery(students, null, null, null,
"name");

        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,

c.getString(c.getColumnIndex(StudentsProvider._ID)) +
                ", " + c.getString(c.getColumnIndex(
StudentsProvider.NAME)) +
                ", " + c.getString(c.getColumnIndex(
StudentsProvider.GRADE)),
                Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }
}
```

Cree un nuevo archivo `StudentsProvider.java` en el paquete `com.example.MyApplication` y el siguiente es el contenido de **src / com.example.MyApplication / StudentsProvider.java** –

```
package com.example.MyApplication;

import java.util.HashMap;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;

import android.database.Cursor;
import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;

import android.net.Uri;
import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {
    static final String PROVIDER_NAME =
"com.example.MyApplication.StudentsProvider";
    static final String URL = "content://" + PROVIDER_NAME +
"/students";
    static final Uri CONTENT_URI = Uri.parse(URL);

    static final String _ID = "_id";
    static final String NAME = "name";
    static final String GRADE = "grade";

    private static HashMap<String, String>
STUDENTS_PROJECTION_MAP;

    static final int STUDENTS = 1;
    static final int STUDENT_ID = 2;

    static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);
        uriMatcher.addURI(PROVIDER_NAME, "students/#",
STUDENT_ID);
    }
}
```

```
/**
 * Database specific constant declarations
 */

private SQLiteDatabase db;
static final String DATABASE_NAME = "College";
static final String STUDENTS_TABLE_NAME = "students";
static final int DATABASE_VERSION = 1;
static final String CREATE_DB_TABLE =
    " CREATE TABLE " + STUDENTS_TABLE_NAME +
    " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
    " name TEXT NOT NULL, " +
    " grade TEXT NOT NULL);";

/**
 * Helper class that actually creates and manages
 * the provider's underlying data repository.
 */

private static class DatabaseHelper extends
SQLiteOpenHelper {
    DatabaseHelper(Context context){
        super(context, DATABASE_NAME, null,
DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " +
STUDENTS_TABLE_NAME);
        onCreate(db);
    }
}

@Override
public boolean onCreate() {
    Context context = getContext();
    DatabaseHelper dbHelper = new DatabaseHelper(context);

    /**
     * Create a write able database which will trigger
its
     * creation if it doesn't already exist.
     */
}
```

```

        db = dbHelper.getWritableDatabase();
        return (db == null)? false:true;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        /**
         * Add a new student record
         */
        long rowID = db.insert(      STUDENTS_TABLE_NAME, "",
values);

        /**
         * If record is added successfully
         */
        if (rowID > 0) {
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI,
rowID);
            getContext().getContentResolver().notifyChange(_uri,
null);
            return _uri;
        }

        throw new SQLException("Failed to add a record into " +
uri);
    }

    @Override
    public Cursor query(Uri uri, String[] projection,
        String selection,String[] selectionArgs, String
sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
        qb.setTables(STUDENTS_TABLE_NAME);

        switch (uriMatcher.match(uri)) {
            case STUDENTS:
                qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
                break;

            case STUDENT_ID:
                qb.appendWhere( _ID + "=" +
uri.getPathSegments().get(1));
                break;

            default:
        }

        if (sortOrder == null || sortOrder == ""){
            /**
             * By default sort on student names
             */

```

```

        sortOrder = NAME;
    }

    Cursor c = qb.query(db,    projection,    selection,
        selectionArgs,null, null, sortOrder);
    /**
     * register to watch a content URI for changes
     */
    c.setNotificationUri(getContext().getContentResolver(),
uri);
    return c;
}

@Override
public int delete(Uri uri, String selection, String[]
selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)){
        case STUDENTS:
            count = db.delete(STUDENTS_TABLE_NAME, selection,
selectionArgs);
            break;

        case STUDENT_ID:
            String id = uri.getPathSegments().get(1);
            count = db.delete( STUDENTS_TABLE_NAME, _ID + "
= " + id +
                (!TextUtils.isEmpty(selection) ? "
AND (" + selection + ')' : ""),
selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI "
+ uri);
    }

    getContext().getContentResolver().notifyChange(uri,
null);
    return count;
}

@Override
public int update(Uri uri, ContentValues values,
String selection, String[] selectionArgs) {
    int count = 0;
    switch (uriMatcher.match(uri)) {
        case STUDENTS:
            count = db.update(STUDENTS_TABLE_NAME, values,
selection, selectionArgs);
            break;
    }
}

```

```

        case STUDENT_ID:
            count = db.update(STUDENTS_TABLE_NAME, values,
                _ID + " = " + uri.getPathSegments().get(1) +
                (!TextUtils.isEmpty(selection) ? "
                    AND (" + selection + ') ' : ""), selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI "
+ uri );
    }

    getContext().getContentResolver().notifyChange(uri,
null);
    return count;
}

@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)){
        /**
         * Get all student records
         */
        case STUDENTS:
            return
"vnd.android.cursor.dir/vnd.example.students";
        /**
         * Get a particular student
         */
        case STUDENT_ID:
            return
"vnd.android.cursor.item/vnd.example.students";
        default:
            throw new IllegalArgumentException("Unsupported
URI: " + uri);
    }
}
}

```

A continuación se muestra el contenido modificado del archivo *AndroidManifest.xml* . Aquí hemos agregado la etiqueta <proveedor ... /> para incluir nuestro proveedor de contenido:

```

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

    <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>

            <provider android:name="StudentsProvider"

android:authorities="com.example.MyApplication.StudentsProvid
er"/>
        </application>
</manifest>

```

A continuación se muestra el contenido del archivo **res / layout / activity_main.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: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="com.example.MyApplication.MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Content provider"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorial"
        android:textColor="#ff87ff09"
        android:textSize="30dp"

```

```
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="Add Name"
    android:layout_below="@+id/editText3"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2"
    android:onClick="onClickAddName"/>

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:layout_alignTop="@+id/editText"
    android:layout_alignLeft="@+id/textView1"
    android:layout_alignStart="@+id/textView1"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:hint="Name"
    android:textColorHint="@android:color/holo_blue_light"
/>

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:layout_below="@+id/editText"
    android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
```

```

        android:layout_alignRight="@+id/editText2"
        android:layout_alignEnd="@+id/editText2"
        android:hint="Grade"
        android:textColorHint="@android:color/holo_blue_bright"
    />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Retrive student"
        android:id="@+id/button"
        android:layout_below="@+id/button2"
        android:layout_alignRight="@+id/editText3"
        android:layout_alignEnd="@+id/editText3"
        android:layout_alignLeft="@+id/button2"
        android:layout_alignStart="@+id/button2"
        android:onClick="onClickRetrieveStudents"/>
</RelativeLayout>

```

Asegúrese de tener el siguiente contenido del archivo **res / values / strings.xml** :

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>;

```

Intentemos ejecutar nuestra **aplicación** **Mi aplicación** modificada que acabamos de crear. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio IDE, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador, tenga paciencia porque puede tomar algún tiempo según la velocidad de su computadora.

Ahora ingresemos el **nombre del** estudiante y el **grado** y finalmente haga clic en el botón **Agregar nombre** , esto agregará el registro del estudiante en la base de datos y mostrará un mensaje en la parte inferior que muestra el URI de ContentProvider junto con el número de registro agregado en la base de datos. Esta operación utiliza nuestro método **insert ()** . Repitamos este proceso para agregar algunos estudiantes más en la base de datos de nuestro proveedor de contenido.

Una vez que haya terminado de agregar registros en la base de datos, ahora es el momento de pedirle a ContentProvider que nos devuelva esos registros, así que hagamos clic en el botón **Recuperar estudiantes** que buscará y mostrará todos los registros uno por uno, según la implementación de nuestro método **query ()** .

Puede escribir actividades contra las operaciones de actualización y eliminación proporcionando funciones de devolución de llamada en el archivo **MainActivity.java** y luego modificar la interfaz de usuario para tener botones para actualizar y eliminar operaciones de la misma manera que lo hemos hecho para agregar y leer operaciones.

De esta manera, puede usar un proveedor de contenido existente como la libreta de direcciones o puede usar el concepto de proveedor de contenido para desarrollar aplicaciones agradables orientadas a bases de datos donde puede realizar todo tipo de operaciones de base de datos como leer, escribir, actualizar y eliminar como se explicó anteriormente en el ejemplo.

Fragmentos

Un **fragmento** es una parte de una actividad que permite un diseño de actividades más modular. No estará mal si decimos que un fragmento es una especie de **subactividad** .

Los siguientes son puntos importantes sobre el fragmento:

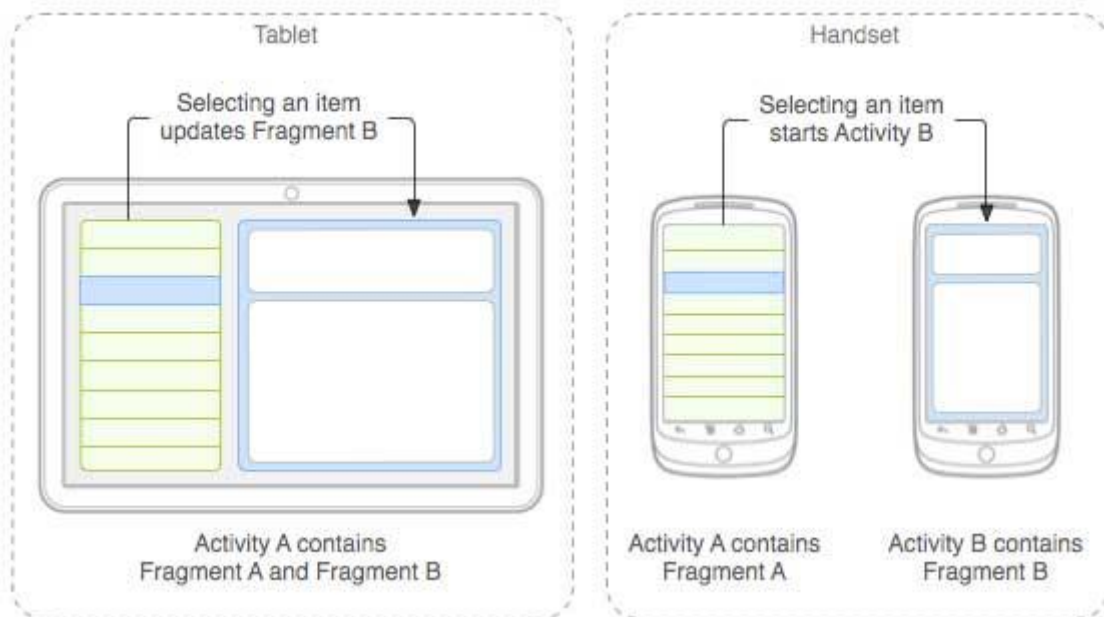
- Un fragmento tiene su propio diseño y su propio comportamiento con sus propias devoluciones de llamada de ciclo de vida.
- Puede agregar o eliminar fragmentos en una actividad mientras la actividad se está ejecutando.
- Puede combinar varios fragmentos en una sola actividad para crear una interfaz de usuario de varios paneles.
- Un fragmento se puede utilizar en múltiples actividades.
- El ciclo de vida de los fragmentos está estrechamente relacionado con el ciclo de vida de la actividad de su anfitrión, lo que significa que cuando la actividad se detiene, todos los fragmentos disponibles en la actividad también se detendrán.
- Un fragmento puede implementar un comportamiento que no tiene ningún componente de interfaz de usuario.
- Se agregaron fragmentos a la API de Android en la versión Honeycomb de Android, cuya versión de API 11.

Puede crear fragmentos extendiendo la clase **Fragment** y puede insertar un fragmento en el diseño de su actividad declarando el fragmento en el archivo de diseño de la actividad, como un elemento **<fragment>** .

Antes de la introducción del fragmento, teníamos una limitación porque solo podíamos mostrar una única actividad en la pantalla en un momento

determinado. Por lo tanto, no pudimos dividir la pantalla del dispositivo y controlar diferentes partes por separado. Pero con la introducción del fragmento obtuvimos más flexibilidad y eliminamos la limitación de tener una sola actividad en la pantalla a la vez. Ahora podemos tener una sola actividad, pero cada actividad puede estar formada por múltiples fragmentos que tendrán su propio diseño, eventos y ciclo de vida completo.

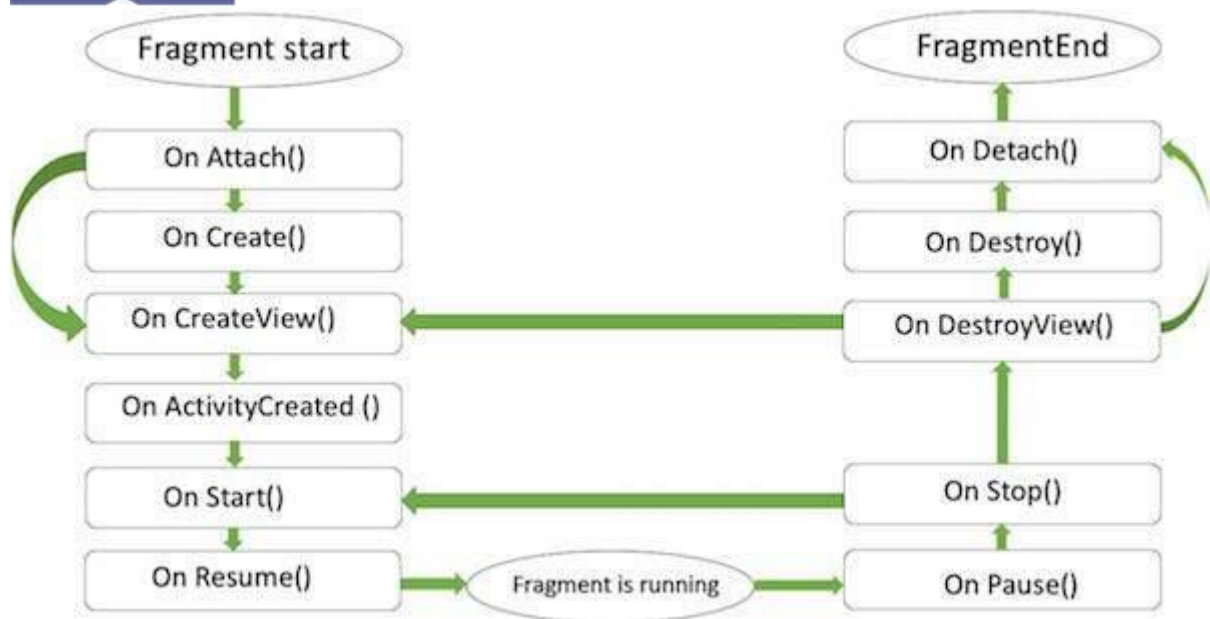
A continuación se muestra un ejemplo típico de cómo dos módulos de UI definidos por fragmentos se pueden combinar en una actividad para el diseño de una tableta, pero separados para el diseño de un teléfono.



La aplicación puede incrustar dos fragmentos en la Actividad A, cuando se ejecuta en un dispositivo del tamaño de una tableta. Sin embargo, en una pantalla del tamaño de un teléfono, no hay suficiente espacio para ambos fragmentos, por lo que la Actividad A incluye solo el fragmento de la lista de artículos, y cuando el usuario selecciona un artículo, inicia la Actividad B, que incluye el segundo fragmento para leer el artículo.

Ciclo de vida del fragmento

Los fragmentos de Android tienen su propio ciclo de vida muy similar a una actividad de Android. Esta sección resume las diferentes etapas de su ciclo de vida.



Ciclo de vida del fragmento

Aquí está la lista de métodos que puede anular en su clase de fragmento:

- **onAttach ()** La instancia del fragmento está asociada con una instancia de actividad. El fragmento y la actividad no están completamente inicializados. Por lo general, obtiene en este método una referencia a la actividad que usa el fragmento para un trabajo de inicialización adicional.
- **onCreate ()** El sistema llama a este método al crear el fragmento. Debe inicializar los componentes esenciales del fragmento que desea retener cuando el fragmento se pausa o se detiene, y luego se reanuda.
- **onCreateView ()** El sistema llama a esta devolución de llamada cuando llega el momento de que el fragmento dibuje su interfaz de usuario por primera vez. Para dibujar una interfaz de usuario para su fragmento, debe devolver un componente de **vista** de este método que es la raíz del diseño de su fragmento. Puede devolver un valor nulo si el fragmento no proporciona una interfaz de usuario.
- **onActivityCreated ()** El `onActivityCreated ()` se llama después del método `onCreateView ()` cuando se crea la actividad del host. Se han creado la actividad y la instancia de fragmento, así como la jerarquía de vista de la actividad. En este punto, se puede acceder a la vista con el método `findViewById ()`. ejemplo. En este método puede crear instancias de objetos que requieren un objeto de contexto
- **onStart ()** El método `onStart ()` se llama una vez que el fragmento se vuelve visible.
- **onResume ()** El fragmento se activa.

- **onPause ()** El sistema llama a este método como la primera indicación de que el usuario está abandonando el fragmento. Normalmente, aquí es donde debe realizar los cambios que deben persistir más allá de la sesión de usuario actual.
- **onStop ()** Fragmento que se detendrá llamando a onStop ()
- **onDestroyView ()** La vista de fragmentos se destruirá después de llamar a este método
- **onDestroy ()** onDestroy () llamado para realizar una limpieza final del estado del fragmento, pero no se garantiza que sea llamado por la plataforma Android.

¿Cómo usar Fragments?

Esto implica una serie de sencillos pasos para crear fragmentos.

- En primer lugar, decida cuántos fragmentos desea utilizar en una actividad. Por ejemplo, queremos usar dos fragmentos para manejar los modos horizontal y vertical del dispositivo.
- A continuación, en función del número de fragmentos, cree clases que ampliarán la clase *Fragmento* . La clase Fragment tiene las funciones de devolución de llamada mencionadas anteriormente. Puede anular cualquiera de las funciones según sus requisitos.
- De acuerdo con cada fragmento, deberá crear archivos de diseño en un archivo XML. Estos archivos tendrán diseño para los fragmentos definidos.
- Finalmente, modifique el archivo de actividad para definir la lógica real de reemplazar fragmentos según sus requisitos.

Tipos de fragmentos

Básicamente, los fragmentos se dividen en tres etapas como se muestra a continuación.

- Fragmento de un solo cuadro: se utilizan para dispositivos de sujeción manual como móviles, aquí solo podemos mostrar un fragmento como vista.
- Fragmentos de listas : los fragmentos que tienen una vista de lista especial se denominan fragmentos de lista
- Transacción de fragmento: uso con transacción de fragmentos. podemos mover un fragmento a otro fragmento.

Fragmentos individuales

Fragmento de un solo cuadro

El fragmento de un solo cuadro está diseñado para dispositivos de pantalla pequeña, como dispositivos de mano (móviles) y debe estar por encima de la versión de Android 3.0.

Ejemplo

Este ejemplo le explicará cómo crear sus propios *Fragmentos*. Aquí crearemos dos fragmentos y uno de ellos se usará cuando el dispositivo esté en modo horizontal y otro fragmento se usará en el caso de modo retrato. Así que sigamos los siguientes pasos de manera similar a lo que seguimos al crear el *ejemplo de Hello World*:

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>MyFragments</i> en un paquete <i>com.example.myfragments</i> , con Activity en blanco.
2	Modifique el archivo de actividad principal <i>MainActivity.java</i> como se muestra a continuación en el código. Aquí verificaremos la orientación del dispositivo y, en consecuencia, cambiaremos entre diferentes fragmentos.
3	Cree dos archivos java <i>PM_Fragment.java</i> y <i>LM_Fragment.java</i> en el paquete <i>com.example.myfragments</i> para definir sus fragmentos y métodos asociados.
4	Cree los archivos de diseños <i>res / layout / lm_fragment.xml</i> y <i>res / layout / pm_fragment.xml</i> y defina sus diseños para ambos fragmentos.
5	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir ambos fragmentos.
6	Defina las constantes requeridas en el archivo <i>res / values / strings.xml</i>
7	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **MainActivity.java**:

```
package com.example.myfragments;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.res.Configuration;
import android.os.Bundle;

public class MainActivity extends Activity {
```

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Configuration config =
getResources().getConfiguration();

    FragmentManager fragmentManager = getFragmentManager();
    FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

    /**
     * Check the device orientation and act accordingly
     */

    if (config.orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
        /**
         * Landscape mode of the device
         */
        LM_Fragment ls_fragment = new LM_Fragment();
        fragmentTransaction.replace(android.R.id.content,
ls_fragment);
    }else{
        /**
         * Portrait mode of the device
         */
        PM_Fragment pm_fragment = new PM_Fragment();
        fragmentTransaction.replace(android.R.id.content,
pm_fragment);
    }
    fragmentTransaction.commit();
}
}
```

Cree dos archivos de fragmentos **LM_Fragment.java** y **PM_Fragment.java**

A continuación se muestra el contenido del archivo **LM_Fragment.java** :

```
package com.example.myfragments;

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

```
public class LM_Fragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        /**
         * Inflate the layout for this fragment
         */
        return inflater.inflate(R.layout.lm_fragment,
            container, false);
    }
}
```

A continuación se muestra el contenido del archivo **PM_Fragment.java** :

```
package com.example.myfragments;

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

public class PM_Fragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        /**
         * Inflate the layout for this fragment
         */
        return inflater.inflate(R.layout.pm_fragment,
            container, false);
    }
}
```

Cree dos archivos de diseño **lm_fragment.xml** y **pm_fragment.xml** en el directorio *res / layout* .

A continuación se muestra el contenido del archivo **lm_fragment.xml** :

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

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#7bae16">

    <TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/landscape_message"
        android:textColor="#000000"
        android:textSize="20px" />

<!-- More GUI components go here -->

</LinearLayout>
```

A continuación se muestra el contenido del archivo **pm_fragment.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#666666">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/portrait_message"
        android:textColor="#000000"
        android:textSize="20px" />

    <!-- More GUI components go here -->

</LinearLayout>
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** que incluye sus fragmentos:

```
<?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="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <fragment
        android:name="com.example.fragments"
        android:id="@+id/lm_fragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
```

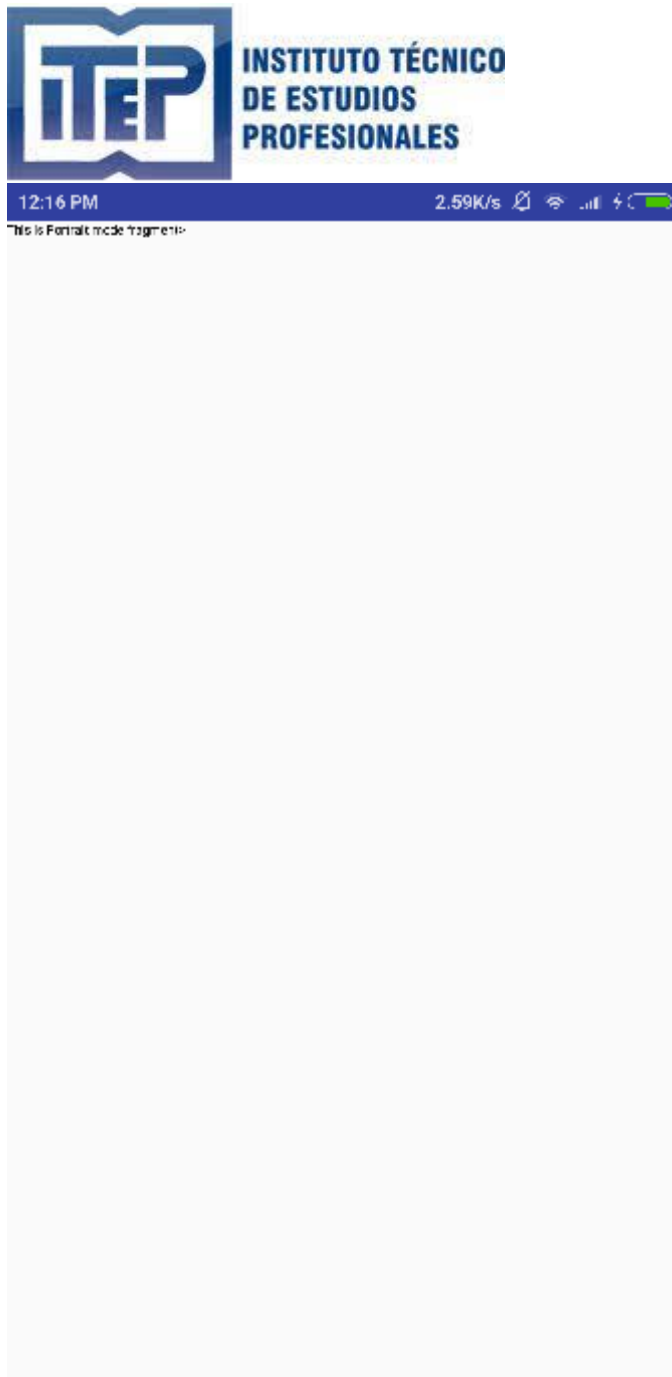
```
<fragment
    android:name="com.example.fragments"
    android:id="@+id/pm_fragment"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />

</LinearLayout>
```

Asegúrese de tener el siguiente contenido del archivo **res / values / strings.xml** –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="landscape_message">This is Landscape mode
fragment</string>
    <string name="portrait_message">This is Portrait mode
fragment</string>
</resources>
```

Intentemos ejecutar nuestra aplicación **MyFragments** modificada que acabamos de crear. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, mostrará la ventana del Emulador donde hará clic en el botón Menú para ver la siguiente ventana. Tenga paciencia porque puede tomar algún tiempo según la velocidad de su computadora.



Para cambiar el modo de la pantalla del emulador, hagamos lo siguiente:

- **fn + control + F11** en Mac para cambiar el paisaje a retrato y viceversa.
- **ctrl + F11** en Windows.
- **ctrl + F11** en Linux.

Una vez que haya cambiado el modo, podrá ver la GUI que ha implementado para el modo horizontal como se muestra a continuación:



De esta manera, puede usar la misma actividad pero diferentes GUI a través de diferentes fragmentos. Puede utilizar diferentes tipos de componentes de GUI para diferentes GUI según sus requisitos.

Fragmento de lista

Versión de soporte de biblioteca estática del ListFragment del marco. Se utiliza para escribir aplicaciones que se ejecutan en plataformas anteriores a Android 3.0. Cuando se ejecuta en Android 3.0 o superior, esta implementación todavía se usa.

La implementación básica del fragmento de lista es para crear una lista de elementos en fragmentos



Lista en fragmentos

Ejemplo

Este ejemplo le explicará cómo crear su propio fragmento de lista basado en `arrayAdapter`. Así que sigamos los siguientes pasos de manera similar a lo que seguimos al crear el ejemplo de Hello World:

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla <i>SimpleListFragment</i> en un paquete <i>com.example.myapplication</i> , con Activity en blanco.
2	Modifique el archivo de cadena, que se ha colocado en <i>res / values / string.xml</i> para agregar nuevas constantes de cadena

- 3 Cree un diseño llamado *list_fragment.xml* en el directorio *res / layout* para definir los fragmentos de su lista. y agregue la etiqueta de fragmento (<fragment>) a su *activity_main.xml*
- 4 Cree un *myListFragment.java*, que se coloca en *java / myListFragment.java* y contiene *onCreateView ()* , *onActivityCreated ()* y *OnItemClickListener ()*
- 5 Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

Antes de comenzar a codificar, inicializaré las constantes de cadena dentro del archivo *string.xml* en el *directorio res / values*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ListFragmentDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="imgdesc">imgdesc</string>

    <string-array name="Planets">
        <item>Sun</item>
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** . contenía diseño lineal y etiqueta de fragmento.

```
<?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" >

    <fragment
        android:id="@+id/fragment1"
        android:name="com.example.myapplication.MyListFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

```
</LinearLayout>
```

A continuación se **muestra** el contenido del archivo **res / layout / list_fragment.xml** . contenía diseño lineal, vista de lista y vista de texto

```
<?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" >

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

    <TextView
        android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </TextView>
</LinearLayout>
```

El siguiente será el contenido del **archivo src / main / java / myListFragment.java**. Antes de escribir en el código, debe seguir algunos pasos como se muestra a continuación

- Cree una clase MyListFragment y extiéndala a ListFragment.
- Dentro del método **onCreateView ()** , infle la vista con el diseño xml list_fragment definido anteriormente.
- Dentro del método **onActivityCreated ()** , cree un arrayadapter a partir del recurso, es decir, utilizando String array R.array.planet que puede encontrar dentro del string.xml y configure este adaptador en listview y también configure el onItem click Listener.
- Dentro del método **OnItemClickListener ()** , muestra un mensaje de brindis con el nombre del elemento en el que se hace clic.

```
package com.example.myapplication;

import android.annotation.SuppressLint;
import android.app.ListFragment;
import android.os.Bundle;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import android.widget.AdapterView;
```

```
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Toast;

public class MyListFragment extends ListFragment implements
OnItemClickListener {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.list_fragment,
container, false);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        ArrayAdapter adapter =
ArrayAdapter.createFromResource(getActivity(),
        R.array.Planets,
android.R.layout.simple_list_item_1);
        setListAdapter(adapter);
        getListView().setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
        Toast.makeText(getActivity(), "Item: " + position,
Toast.LENGTH_SHORT).show();
    }
}
```

El siguiente código será el contenido de MainActivity.java

```
package com.example.myapplication;

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);
    }
}
```

El siguiente código será el contenido de manifest.xml, que se ha colocado en res / AndroidManifest.xml

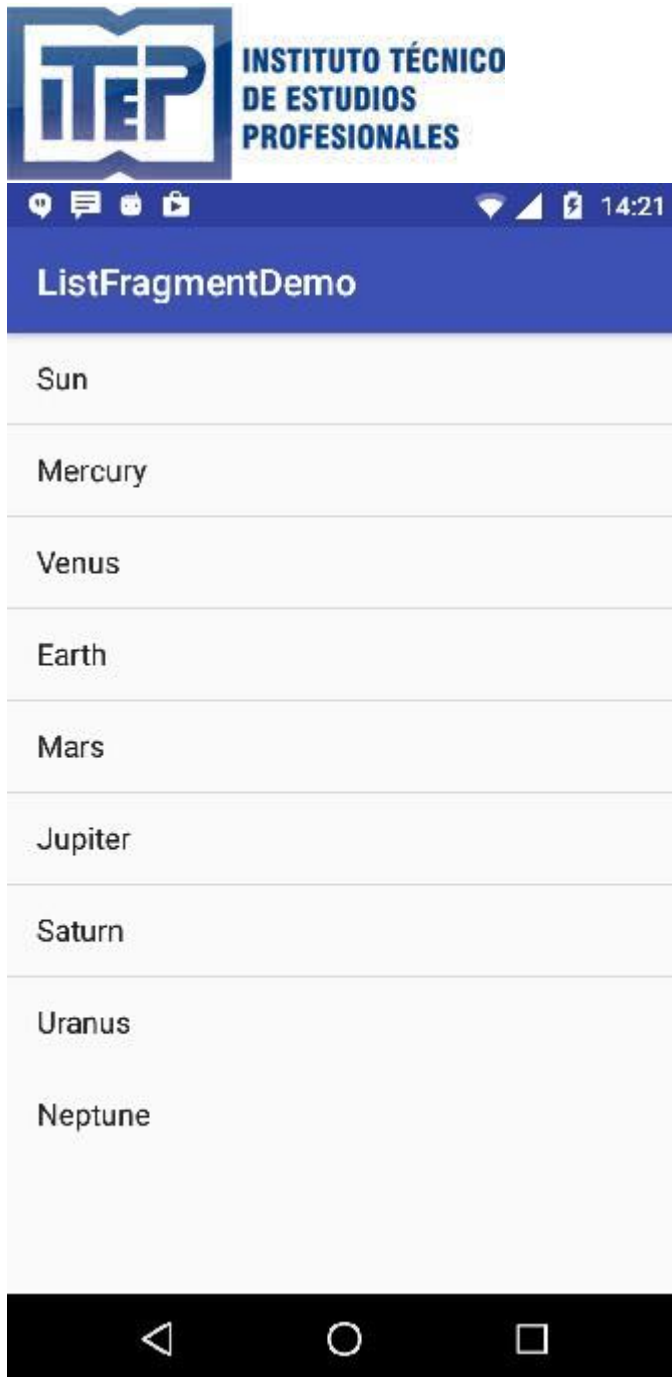
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <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>
```

Ejecutando la Aplicación

Intentemos ejecutar nuestra aplicación **SimpleListFragment** que acabamos de crear. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:



Transición de fragmentos

¿Qué es una transición?

Las transiciones de actividad y fragmentos en Lollipop se basan en una función relativamente nueva en Android llamada Transiciones. Introducido en KitKat, el marco de transición proporciona una API conveniente para animar entre diferentes estados de la interfaz de usuario en una aplicación. El marco se basa

en dos conceptos clave: escenas y transiciones. Una escena define un estado dado de la interfaz de usuario de una aplicación, mientras que una transición define el cambio animado entre dos escenas.

Cuando cambia una escena, una transición tiene dos responsabilidades principales:

- Capture el estado de cada vista tanto en la escena inicial como en la final.
- Crea un animador basado en las diferencias que animarán las vistas de una escena a otra.

Ejemplo

Este ejemplo le explicará cómo crear su animación personalizada con transición de fragmentos. Así que sigamos los siguientes pasos de manera similar a lo que seguimos al crear el ejemplo de Hello World:

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla como <i>fragmentcustomanimations</i> en un paquete <i>com.example.fragmentcustomanimations</i> , con Activity en blanco.
2	Modifique <i>activity_main.xml</i> , que se ha colocado en <i>res / layout / activity_main.xml</i> para agregar una Vista de texto
3	Cree un diseño llamado <i>fragment_stack.xml.xml</i> en el directorio <i>res / layout</i> para definir su etiqueta de fragmento y etiqueta de botón
4	Cree una carpeta, que se coloca en <i>res / y asígnele el nombre de animación</i> y agregue <i>fragment_slide_right_enter.xml</i> <i>fragment_slide_left_exit.xml</i> <i>fragment_slide_right_exit.xml</i> y <i>fragment_slide_left_enter.xml</i>
5	En <i>MainActivity.java</i> , es necesario agregar la pila de fragmentos, el administrador de fragmentos y <i>onCreateView ()</i>
6	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

el siguiente será el contenido de *res.layout / activity_main.xml* que contenía *TextView*

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_vertical|center_horizontal"
```



```
        android:text="@string/hello_world"

        android:textAppearance="?android:attr/textAppearanceMedium"
    />

```

A continuación se muestra el contenido del archivo **res / animation / fragment_stack.xml** . contenía diseño de marco y botón

```
<?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" >

    <fragment
        android:id="@+id/fragment1"

        android:name="com.pavan.listfragmentdemo.MyListFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>

```

A continuación se muestra el contenido del archivo **res / animation / fragment_slide_left_enter.xml** . contenía un método de conjunto y un animador de objetos

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

        android:interpolator="@android:interpolator/decelerate_quint"
        android:valueFrom="100dp" android:valueTo="0dp"
        android:valueType="floatType"
        android:propertyName="translationX"

        android:duration="@android:integer/config_mediumAnimTime" />

    <objectAnimator

        android:interpolator="@android:interpolator/decelerate_quint"
        android:valueFrom="0.0" android:valueTo="1.0"
        android:valueType="floatType"
        android:propertyName="alpha"

        android:duration="@android:integer/config_mediumAnimTime" />
    </set>

```

El siguiente será el contenido de **res / animation / fragment_slide_left_exit.xml** file.it contiene etiquetas de animador de objetos y conjuntos.

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

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="0dp" android:valueTo="-100dp"
    android:valueType="floatType"
    android:propertyName="translationX"

android:duration="@android:integer/config_mediumAnimTime" />

    <objectAnimator

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="1.0" android:valueTo="0.0"
    android:valueType="floatType"
    android:propertyName="alpha"

android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

El siguiente código será el contenido de **res / animation / fragment_slide_right_enter.xml** file.it contiene etiquetas de animador de objetos y conjuntos

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

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="-100dp" android:valueTo="0dp"
    android:valueType="floatType"
    android:propertyName="translationX"

android:duration="@android:integer/config_mediumAnimTime" />

    <objectAnimator

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="0.0" android:valueTo="1.0"
    android:valueType="floatType"
    android:propertyName="alpha"

android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

El siguiente código será el contenido del archivo **res / animation / fragment_slide_right_exit.xml**, que contenía etiquetas de animador de objetos y conjuntos

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

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="0dp" android:valueTo="100dp"
    android:valueType="floatType"
    android:propertyName="translationX"

android:duration="@android:integer/config_mediumAnimTime" />

    <objectAnimator

android:interpolator="@android:interpolator/decelerate_quint"
    android:valueFrom="1.0" android:valueTo="0.0"
    android:valueType="floatType"
    android:propertyName="alpha"

android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

El siguiente código será el contenido del archivo **src / main / java / MainActivity.java**. contenía oyente de botón, fragmento de pila y onCreateView

```
package com.example.fragmentcustomanimations;

import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentTransaction;
import android.os.Bundle;

import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;

import android.widget.Button;
import android.widget.TextView;

/**
 * Demonstrates the use of custom animations in a
 * FragmentTransaction when
 * pushing and popping a stack.
 */
public class FragmentCustomAnimations extends Activity {
```

```
int mStackLevel = 1;

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

    // Watch for button clicks.
    Button button =
(Button)findViewById(R.id.new_fragment);

    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            addFragmentToStack();
        }
    });

    if (savedInstanceState == null) {
        // Do first time initialization -- add initial
fragment.
        Fragment newFragment =
CountingFragment.newInstance(mStackLevel);
        FragmentTransaction ft =
getFragmentManager().beginTransaction();
        ft.add(R.id.simple_fragment, newFragment).commit();
    }
    else
    {
        mStackLevel = savedInstanceState.getInt("level");
    }
}

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("level", mStackLevel);
}

void addFragmentToStack() {

    mStackLevel++;

    // Instantiate a new fragment.
    Fragment newFragment =
CountingFragment.newInstance(mStackLevel);

    // Add the fragment to the activity, pushing this
transaction
    // on to the back stack.
    FragmentTransaction ft =
getFragmentManager().beginTransaction();
```

```
ft.setCustomAnimations(R.animator.fragment_slide_left_enter,
    R.animator.fragment_slide_left_exit,
    R.animator.fragment_slide_right_enter,
    R.animator.fragment_slide_right_exit);
ft.replace(R.id.simple_fragment, newFragment);
ft.addToBackStack(null);
ft.commit();
}

public static class CountingFragment extends Fragment {
    int mNum;
    /**
     * Create a new instance of CountingFragment, providing
     "num"
     * as an argument.
     */
    static CountingFragment newInstance(int num) {
        CountingFragment f = new CountingFragment();

        // Supply num input as an argument.
        Bundle args = new Bundle();
        args.putInt("num", num);
        f.setArguments(args);
        return f;
    }

    /**
     * When creating, retrieve this instance's number from
     its arguments.
     */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mNum = getArguments() != null ?
getArguments().getInt("num") : 1;
    }
    /**
     * The Fragment's UI is just a simple text view showing
     its
     * instance number.
     */

    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.hello_world,
container, false);
        View tv = v.findViewById(R.id.text);
        ((TextView)tv).setText("Fragment #" + mNum);
    }
}
```

```
        tv.setBackgroundDrawable(getResources().  
            getDrawable(android.R.drawable.gallery_thumb));  
        return v;  
    }  
}
```

el siguiente será el contenido de **AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>  
    <manifest  
        xmlns:android="http://schemas.android.com/apk/res/android"  
        package="com.example.fragmentcustomanimations"  
        android:versionCode="1"  
        android:versionName="1.0" >  
  
        <application  
            android:allowBackup="true"  
            android:icon="@drawable/ic_launcher"  
            android:label="@string/app_name"  
            android:theme="@style/AppTheme" >  
  
            <activity  
  
                android:name="com.example.fragmentcustomanimations.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>  
  
            </application>  
        </manifest>
```

Ejecutando la Aplicación

Intentemos ejecutar nuestra aplicación **Fragment Transitions** que acabamos de crear. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, mostrará la siguiente ventana del emulador:



Si hace clic en un nuevo fragmento, se cambiará el primer fragmento al segundo fragmento como se muestra a continuación



Intents and Filters

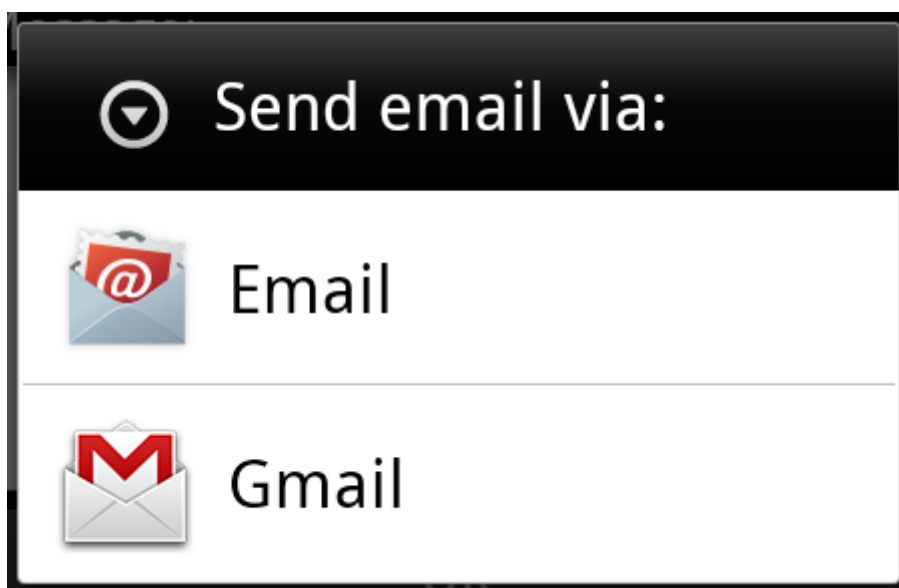
Un **intent** de Android es una descripción abstracta de una operación que se realizará. Se puede usar con **startActivity** para iniciar una actividad, **broadcastIntent** para enviarla a cualquier componente de BroadcastReceiver interesado y **startService (Intent)** o **bindService (Intent, ServiceConnection, int)** para comunicarse con un servicio en segundo plano.

Intent en sí, un objeto intent, es una estructura de datos pasiva que contiene una descripción abstracta de una operación a realizar.

Por ejemplo, supongamos que tiene una actividad que necesita iniciar un cliente de correo electrónico y envía un correo electrónico usando su dispositivo Android. Para este propósito, tu Actividad enviaría un ACTION_SEND junto con el **selector** apropiado al Resolver Intent de Android. El selector especificado proporciona la interfaz adecuada para que el usuario elija cómo enviar sus datos de correo electrónico.

```
Intent email = new Intent(Intent.ACTION_SEND,
Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT,
subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email, "Choose an email
client from..."));
```

La sintaxis anterior está llamando al método startActivity para iniciar una actividad de correo electrónico y el resultado debe ser el que se muestra a continuación:



Por ejemplo, suponga que tiene una actividad que necesita abrir una URL en un navegador web en su dispositivo Android. Para este propósito, su Actividad enviará ACTION_WEB_SEARCH Intent al Intent Resolver de Android para abrir la URL dada en el navegador web. Intent Resolver analiza una lista de actividades y elige la que mejor se adapte a su intención, en este caso, la actividad del navegador web. Intent Resolver luego pasa su página web al navegador web e inicia la actividad del navegador web.

```
String q = "tutorial";
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );
intent.putExtra(SearchManager.QUERY, q);
startActivity(intent);
```

El ejemplo anterior buscará como **tutorial** en el motor de búsqueda de Android y dará el resultado de tutorial en su actividad

Existen mecanismos separados para entregar intenciones a cada tipo de componente: actividades, servicios y receptores de transmisión.

No Señor	Método y descripción
1	Context.startActivity () El objeto Intent se pasa a este método para iniciar una nueva actividad u obtener una actividad existente para hacer algo nuevo.
2	Context.startService () El objeto Intent se pasa a este método para iniciar un servicio o entregar nuevas instrucciones a un servicio en curso.
3	Context.sendBroadcast () El objeto Intent se pasa a este método para entregar el mensaje a todos los receptores de difusión interesados.

Objetos intent

Un objeto Intent es un conjunto de información que utiliza el componente que recibe la intención, así como la información utilizada por el sistema Android.

Un objeto Intent puede contener los siguientes componentes en función de lo que comunica o va a realizar:

Action

Esta es una parte obligatoria del objeto Intent y es una cadena que nombra la acción a realizar o, en el caso de las intenciones de difusión, la acción que tuvo lugar y se informa. La acción determina en gran medida cómo se estructura el resto del objeto de intención. La clase Intent define una serie de constantes de acción correspondientes a diferentes intenciones.

La acción en un objeto Intent se puede establecer mediante el método `setAction ()` y leer mediante `getAction ()`.

Data

Agrega una especificación de datos a un filtro de intención. La especificación puede ser solo un tipo de datos (el atributo `mimeType`), solo un URI o tanto un tipo de datos como un URI. Un URI se especifica mediante atributos separados para cada una de sus partes:

Estos atributos que especifican el formato de URL son opcionales, pero también dependen mutuamente:

- Si no se especifica un esquema para el filtro de intención, se ignoran todos los demás atributos de URI.
- Si no se especifica un host para el filtro, se ignoran el atributo del puerto y todos los atributos de la ruta.

El método `setData ()` especifica datos solo como un URI, `setType ()` los especifica solo como un tipo MIME y `setDataAndType ()` los especifica como un URI y un tipo MIME. El URI es leído por `getData ()` y el tipo por `getType ()`.

Algunos ejemplos de pares de acción / datos son:

No Señor.	Acción / Par de datos y descripción
1	Contenido de ACTION_VIEW: // contactos / personas / 1 Muestra información sobre la persona cuyo identificador es "1".
2	Contenido de ACTION_DIAL: // contactos / personas / 1 Muestre el marcador telefónico con la persona completada.
3	ACTION_VIEW tel: 123 Muestre el marcador telefónico con el número indicado.
4	ACTION_DIAL tel: 123 Muestre el marcador telefónico con el número indicado.
5	ACTION_EDIT: content: // contactos / personas / 1 Edite la información sobre la persona cuyo identificador es "1".
6	ACTION_VIEW content: // contactos / personas / Muestra una lista de personas, por la que el usuario puede navegar.
7	ACTION_SET_WALLPAPER Mostrar configuraciones para elegir fondo de pantalla
8	ACTION_SYNC Será sincrónico con los datos, el valor constante es android.intent.action.SYNC
9	ACTION_SYSTEM_TUTORIAL Comenzará el tutorial definido por la plataforma (tutorial predeterminado o tutorial de inicio)
10	ACTION_TIMEZONE_CHANGED

	Da a entender cuando la zona horaria ha cambiado
11	ACTION_UNINSTALL_PACKAGE Se utiliza para ejecutar el desinstalador predeterminado.

Categoría

La categoría es una parte opcional del objeto Intent y es una cadena que contiene información adicional sobre el tipo de componente que debe manejar la intención. El método `addCategory ()` coloca una categoría en un objeto Intent, `removeCategory ()` elimina una categoría previamente agregada y `getCategories ()` obtiene el conjunto de todas las categorías actualmente en el objeto.

Puede consultar los detalles de los filtros de intención en la siguiente sección para comprender cómo usamos las categorías para elegir la actividad apropiada correspondiente a una intención.

Extras

Esto estará en pares clave-valor para obtener información adicional que debe entregarse al componente que maneja la intención. Los extras se pueden configurar y leer usando los métodos `putExtras ()` y `getExtras ()` respectivamente.

Flags

Estas flags son parte opcional del objeto Intent e indican al sistema Android cómo iniciar una actividad y cómo tratarla después de que se inicie, etc.

No Señor	Banderas y descripción
1	FLAG_ACTIVITY_CLEAR_TASK Si se establece en un Intent pasado a <code>Context.startActivity ()</code> , este indicador hará que cualquier tarea existente que esté asociada con la actividad se borre antes de que se inicie la actividad. Es decir, la actividad se convierte en la nueva raíz de una tarea que de otro modo estaría vacía y todas las actividades anteriores se terminan. Esto solo se puede usar junto con <code>FLAG_ACTIVITY_NEW_TASK</code> .
2	FLAG_ACTIVITY_CLEAR_TOP Si se establece, y la actividad que se está iniciando ya se está ejecutando en la tarea actual, entonces, en lugar de lanzar una nueva instancia de esa actividad, todas las demás actividades encima de ella se cerrarán y este Intent se entregará al (ahora en arriba) la actividad anterior como una nueva intención.
3	FLAG_ACTIVITY_NEW_TASK

Esta bandera es utilizada generalmente por actividades que quieren presentar un comportamiento de estilo "lanzador": le dan al usuario una lista de cosas separadas que se pueden hacer, que de otra manera se ejecutan completamente independientemente de la actividad que las lanza.

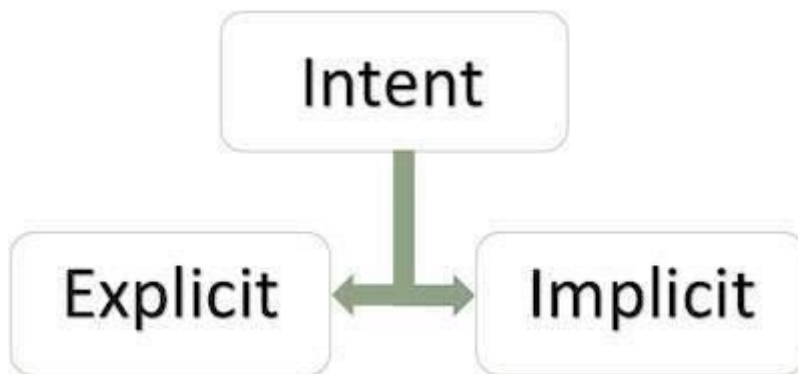
ComponentName

Este campo opcional es un objeto **ComponentName** de Android que representa la clase Activity, Service o BroadcastReceiver. Si está configurado, el objeto Intent se entrega a una instancia de la clase designada; de lo contrario, Android usa otra información en el objeto Intent para localizar un objetivo adecuado.

El nombre del componente lo establece `setComponent ()`, `setClass ()` o `setClassName ()` y lo lee `getComponent ()`.

Tipos de intent

Hay dos tipos de intents compatibles con Android



Intenciones explícitas

La intención explícita va a estar conectada al mundo interno de la aplicación, supongamos que si desea conectar una actividad a otra actividad, podemos hacer esta cita con una intención explícita, la imagen de abajo conecta la primera actividad a la segunda actividad haciendo clic en el botón.



Estas intenciones designan el componente de destino por su nombre y generalmente se usan para mensajes internos de la aplicación, como una actividad que inicia un servicio subordinado o una actividad hermana. Por ejemplo

```
// Explicit Intent by specifying its class name
Intent i = new Intent(FirstActivity.this,
    SecondActivity.class);

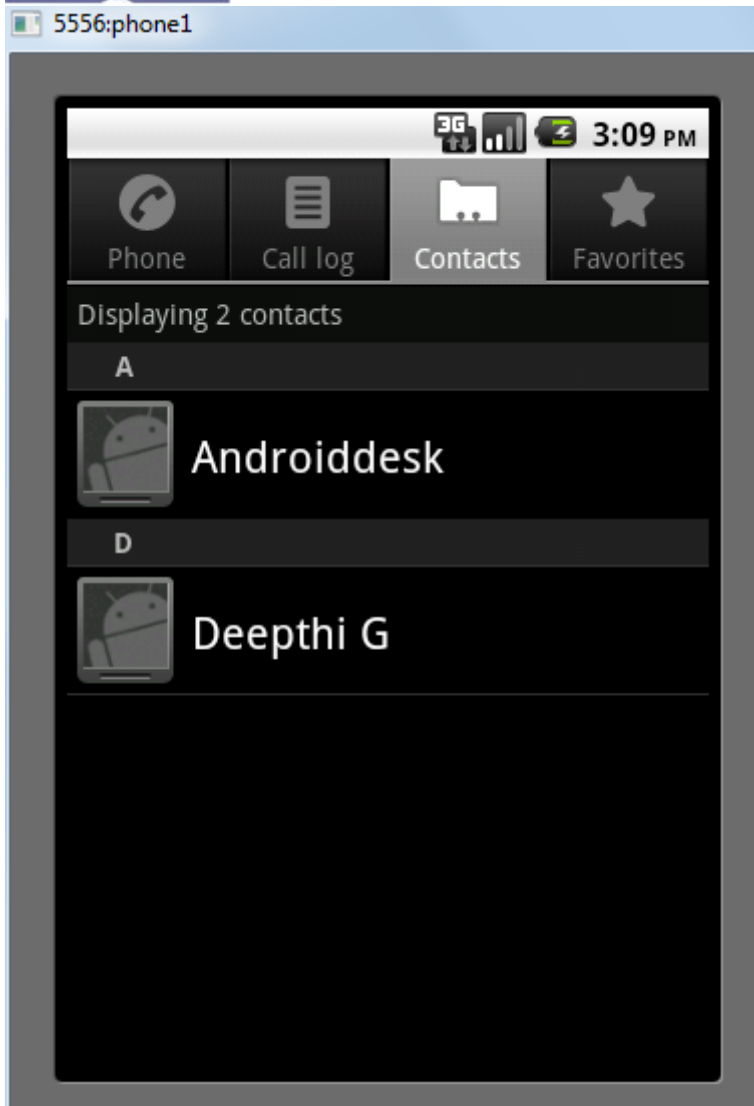
// Starts TargetActivity
startActivity(i);
```

Intenciones implícitas

Estas intenciones no nombran un objetivo y el campo para el nombre del componente se deja en blanco. Las intenciones implícitas se utilizan a menudo para activar componentes en otras aplicaciones. Por ejemplo

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

El código anterior dará el resultado que se muestra a continuación



El componente de destino que recibe la intención puede usar el método **getExtras ()** para obtener los datos adicionales enviados por el componente de origen. Por ejemplo

```
// Get bundle object at appropriate place in your code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Ejemplo

El siguiente ejemplo muestra la funcionalidad de un intento de Android para iniciar varias aplicaciones integradas de Android.

Paso

Descripción

1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>Mi aplicación</i> en un paquete <i>com.example.myapplication</i> .
2	Modifique el archivo <i>src / main / java / MainActivity.java</i> y agregue el código para definir dos oyentes correspondientes a dos botones, es decir. Inicie el navegador y el teléfono de inicio.
3	Modifique el archivo XML de diseño <i>res / layout / activity_main.xml</i> para agregar tres botones en diseño lineal.
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.My Application / MainActivity.java** .

```
package com.example.myapplication;

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 MainActivity extends AppCompatActivity {
    Button b1,b2;

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

        b1=(Button) findViewById(R.id.button);
        b1.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                    Uri.parse("http://www.example.com"));
                startActivity(i);
            }
        });

        b2=(Button) findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```

        Intent i = new
Intent(android.content.Intent.ACTION_VIEW,
        Uri.parse("tel:9510300000"));
        startActivity(i);
    }
});
}
}

```

A continuación se muestra el contenido del archivo **res / layout / activity_main.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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Intent Example"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorial"
        android:textColor="#ff87ff09"
        android:textSize="30dp"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true" />

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageButton"
        android:src="@drawable/abc"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true" />

```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Browser"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Phone"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2" />
</RelativeLayout>
```

A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Applicaiton</string>
</resources>
```

A continuación se muestra el contenido predeterminado de **AndroidManifest.xml** :


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

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

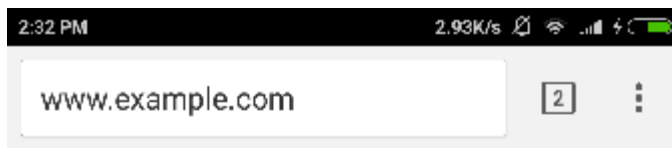
```

        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>

```

Intentemos ejecutar su **aplicación** Mi aplicación. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar de la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, se mostrará el siguiente Emulador ventana -

Ahora haga clic en el botón **Iniciar navegador** , que iniciará un navegador configurado y mostrará <http://www.example.com> como se muestra a continuación:



Example Domain

This domain is established to be used for illustrative examples in documents. You may use this domain in examples without prior coordination or asking for permission.

[More information...](#)

De manera similar, puede iniciar la interfaz del teléfono usando el botón Iniciar teléfono, que le permitirá marcar el número de teléfono ya dado.

Intent filters

Has visto cómo se ha utilizado un Intent para llamar a otra actividad. El sistema operativo Android utiliza filtros para identificar el conjunto de actividades, servicios y receptores de difusión que pueden manejar el intento con la ayuda de un conjunto específico de acciones, categorías y esquema de datos asociados con un intento. Utilizará el elemento **<intent-filter>** en el archivo de manifiesto para enumerar acciones, categorías y tipos de datos asociados con cualquier actividad, servicio o receptor de transmisión.

A continuación se muestra un ejemplo de una parte del archivo **AndroidManifest.xml** para especificar una actividad **com.example.My Application.CustomActivity** que puede ser invocada por cualquiera de las dos acciones mencionadas, una categoría y un dato:

```
<activity android:name=".CustomActivity"
    android:label="@string/app_name">

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="com.example.My
Application.LAUNCH" />
        <category
android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>

</activity>
```

Una vez que esta actividad se define junto con los filtros mencionados anteriormente, otras actividades podrán invocar esta actividad usando **android.intent.action.VIEW**, o usando la acción **com.example.My Application.LAUNCH** siempre que su categoría sea **android.intent.category.DEFAULT**.

El elemento **<data>** especifica el tipo de datos que se espera que se llame a la actividad y, para el ejemplo anterior, nuestra actividad personalizada espera que los datos comiencen con "http: //"

Puede haber una situación en la que una intención pueda pasar por los filtros de más de una actividad o servicio, se le puede preguntar al usuario qué componente activar. Se genera una excepción si no se puede encontrar ningún objetivo.

Existen las siguientes comprobaciones de prueba de Android antes de invocar una actividad:

- Un filtro **<intent-filter>** puede enumerar más de una acción como se muestra arriba, pero esta lista no puede estar vacía; un filtro debe contener al menos un elemento **<action>**, de lo contrario bloqueará todas las intenciones. Si se menciona más de una acción, Android intentará

hacer coincidir una de las acciones mencionadas antes de invocar la actividad.

- Un filtro `<intent-filter>` puede incluir cero, una o más de una categoría. Si no se menciona ninguna categoría, Android siempre pasa esta prueba, pero si se menciona más de una categoría, entonces para que la intención pase la prueba de categoría, cada categoría en el objeto Intent debe coincidir con una categoría en el filtro.
- Cada elemento `<data>` puede especificar un URI y un tipo de datos (tipo de medio MIME). Hay atributos separados como **esquema**, **host**, **puerto** y **ruta** para cada parte del URI. Un objeto Intent que contiene tanto un URI como un tipo de datos pasa la parte del tipo de datos de la prueba solo si su tipo coincide con un tipo enumerado en el filtro.

Ejemplo

El siguiente ejemplo es una modificación del ejemplo anterior. Aquí veremos cómo Android resuelve el conflicto si una intención invoca dos actividades definidas en, a continuación, cómo invocar una actividad personalizada usando un filtro y la tercera es un caso de excepción si Android no presenta la actividad adecuada definida para una intención.

Paso	Descripción
1	Utilizará android studio para crear una aplicación de Android y nombrarla como <i>Mi aplicación</i> en paquete <i>com.example.myapplication</i> ; .
2	Modifique el archivo <i>src / Main / Java / MainActivity.java</i> y agregue el código para definir tres oy correspondientes a tres botones definidos en el archivo de diseño.
3	Agregue un nuevo archivo <i>src / Main / Java / CustomActivity.java</i> para tener una actividad perso invocada por diferentes intenciones.
4	Modifique el archivo XML de diseño <i>res / layout / activity_main.xml</i> para agregar tres botones en
5	Agregue un archivo XML de diseño <i>res / layout / custom_view.xml</i> para agregar un <code><TextView></code> los datos pasados a través de la intención.
6	Modifique <i>AndroidManifest.xml</i> para agregar <code><intent-filter></code> a fin de definir reglas para su intencio actividad personalizada.

- | | |
|---|---|
| 7 | Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios aplicación. |
|---|---|

A continuación se muestra el contenido del archivo de actividad principal modificado **src / MainActivity.java** .

```
package com.example.myapplication;

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 MainActivity extends AppCompatActivity {
    Button b1,b2,b3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1=(Button)findViewById(R.id.button);
        b1.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                    Uri.parse("http://www.example.com"));
                startActivity(i);
            }
        });

        b2 = (Button)findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent("com.example
                LAUNCH",Uri.parse("http://www.example.com"));
                startActivity(i);
            }
        });

        b3 = (Button)findViewById(R.id.button3);
        b3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```

        Intent i = new Intent("com.example.
            My Application.LAUNCH",
                Uri.parse("https://www.example.com"));
        startActivity(i);
    }
});
}
}

```

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.My Application / CustomActivity.java**.

```

package com.example.tutorial.myapplication;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class CustomActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.custom_view);
        TextView label = (TextView)
        findViewById(R.id.show_data);
        Uri url = getIntent().getData();
        label.setText(url.toString());
    }
}

```

A continuación se muestra el contenido del archivo **res / layout / activity_main.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: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="com.example.myapplication.MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Intent Example"
        android:layout_alignParentTop="true"

```

```
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorial"
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Browser"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start browsing with launch action"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
    android:layout_alignEnd="@+id/button" />

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

```
        android:text="Exceptional condition"
        android:id="@+id/button3"
        android:layout_below="@+id/button2"
        android:layout_alignLeft="@+id/button2"
        android:layout_alignStart="@+id/button2"
        android:layout_toStartOf="@+id/editText"
        android:layout_alignParentEnd="true" />
</RelativeLayout>
```

A continuación se muestra el contenido del archivo **res / layout / custom_view.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/show_data"
        android:layout_width="fill_parent"
        android:layout_height="400dp"/>
</LinearLayout>
```

A continuación se muestra el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>
```

A continuación se muestra el contenido predeterminado de **AndroidManifest.xml** :

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

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


```
</activity>

<activity
android:name="com.example.myapplication.CustomActivity">

    <intent-filter>
        <action android:name =
"android.intent.action.VIEW" />
        <action android:name =
"com.example.tutorial.myapplication.LAUNCH" />
        <category android:name =
"android.intent.category.DEFAULT" />
        <data android:scheme = "http" />
    </intent-filter>

</activity>
</application>

</manifest>
```

Intentemos ejecutar su **aplicación** Mi aplicación. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:

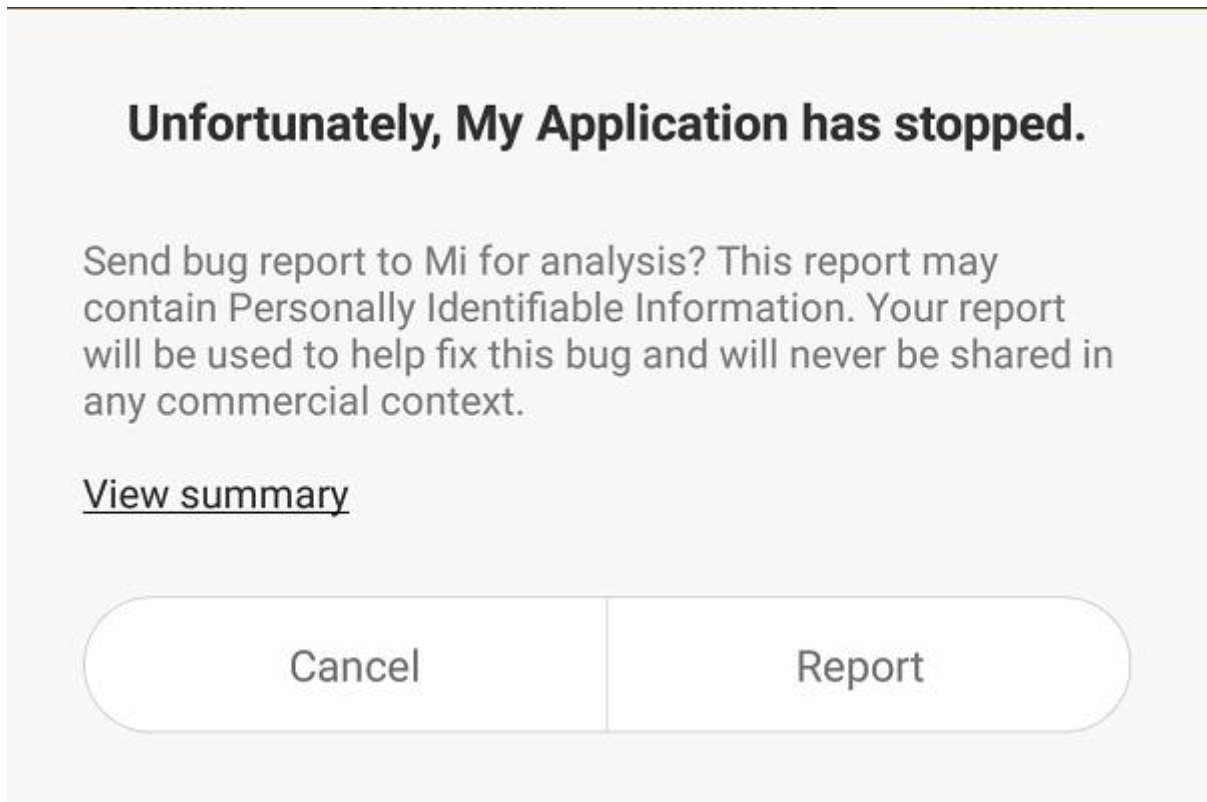
Ahora comencemos con el primer botón "Iniciar navegador con acción VER". Aquí hemos definido nuestra actividad personalizada con un filtro "android.intent.action.VIEW", y ya hay una actividad predeterminada contra la acción VIEW definida por Android que está iniciando el navegador web, por lo que Android muestra las siguientes dos opciones para seleccionar la actividad que quiere lanzar.

Ahora, si selecciona Navegador, Android iniciará el navegador web y abrirá el sitio web example.com, pero si selecciona la opción IndentDemo, Android iniciará CustomActivity, que no hace más que capturar los datos pasados y se muestra en una vista de texto de la siguiente manera:

Ahora regrese usando el botón Atrás y haga clic en el botón "Iniciar navegador con acción de INICIO", aquí Android aplica el filtro para elegir definir actividad y simplemente inicia su actividad personalizada

Nuevamente, regrese usando el botón Atrás y haga clic en el botón "Condición de excepción", aquí Android intenta encontrar un filtro válido para la intención dada pero no encuentra una actividad válida definida porque esta vez usamos

datos como **https** en lugar de **http** aunque estamos dando una acción correcta, por lo que Android genera una excepción y muestra la siguiente pantalla:



Uso de interfaces

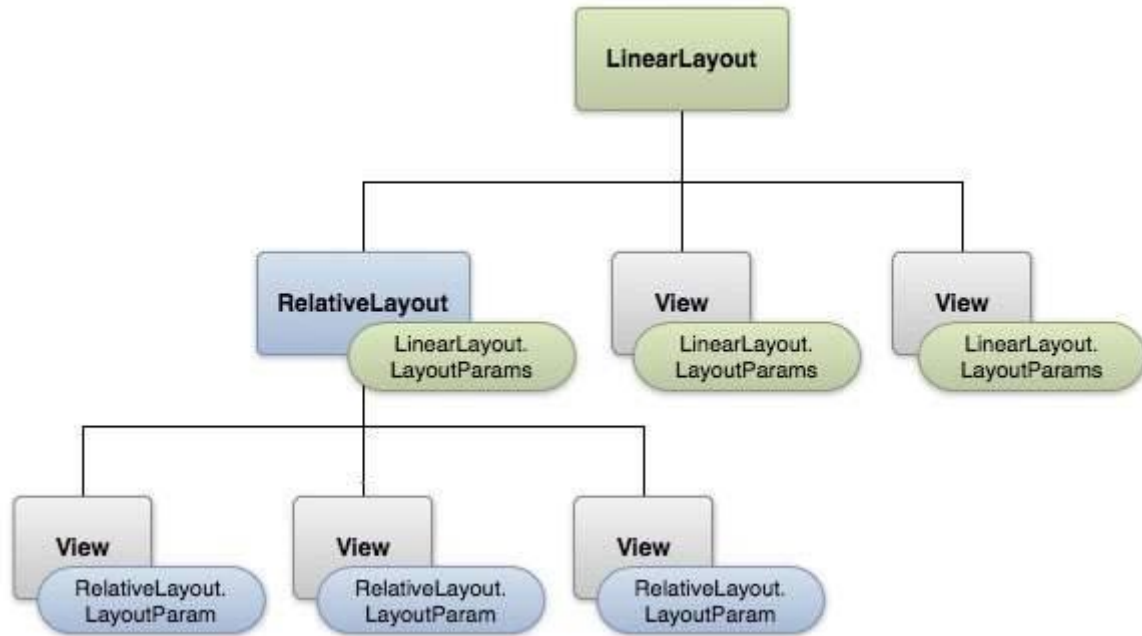
diseños de interfaz de usuario

El bloque de construcción básico para la interfaz de usuario es un objeto **Vista** que se crea a partir de la clase `View` y ocupa un área rectangular en la pantalla y es responsable del dibujo y manejo de eventos. `View` es la clase base de los widgets, que se utilizan para crear componentes de IU interactivos como botones, campos de texto, etc.

El **ViewGroup** es una subclase de **Vista** y proporciona contenedores invisibles que sostienen otras vistas u otros ViewGroups y definir sus propiedades de diseño.

En el tercer nivel, tenemos diferentes diseños que son subclases de la clase `ViewGroup` y un diseño típico define la estructura visual para una interfaz de usuario de Android y se puede crear en tiempo de ejecución

usando objetos **View** / **ViewGroup** o puede declarar su diseño usando un simple archivo XML **main_layout .xml** que se encuentra en la carpeta **res / layout** de su proyecto.



Parámetros de diseño

Este tutorial trata más sobre la creación de su GUI basada en diseños definidos en un archivo XML. Un diseño puede contener cualquier tipo de widgets, como botones, etiquetas, cuadros de texto, etc. A continuación se muestra un ejemplo simple de archivo XML con LinearLayout:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a TextView" />

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a Button" />

    <!-- More GUI components go here -->

```



```
</LinearLayout>
```

Una vez que se haya creado su diseño, puede cargar el recurso de diseño desde el código de su aplicación, en su implementación de devolución de llamada *Activity.onCreate ()* como se muestra a continuación:

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

Tipos de diseño de Android

Hay una serie de diseños proporcionados por Android que utilizará en casi todas las aplicaciones de Android para proporcionar una vista, apariencia y sensación diferentes.

No Señor	Diseño y descripción
1	<p>Linear layout</p> <p>LinearLayout es un grupo de vistas que alinea a todos los elementos secundarios en una sola dirección, vertical u horizontalmente.</p>
2	<p>Relative layout</p> <p>RelativeLayout es un grupo de vistas que muestra vistas secundarias en posiciones relativas.</p>
3	<p>Table layout</p> <p>TableLayout es una vista que agrupa las vistas en filas y columnas.</p>
4	<p>Absolute layout</p> <p>AbsoluteLayout le permite especificar la ubicación exacta de sus hijos.</p>
5	<p>Frame layout</p> <p>FrameLayout es un marcador de posición en la pantalla que puede usar para mostrar una sola vista.</p>
6	<p>List view</p> <p>ListView es un grupo de vistas que muestra una lista de elementos desplazables.</p>
7	<p>Grid view</p> <p>GridView es un ViewGroup que muestra elementos en una cuadrícula desplazable bidimensional.</p>

Atributos de diseño

Cada diseño tiene un conjunto de atributos que definen las propiedades visuales de ese diseño. Hay algunos atributos comunes entre todos los diseños y hay otros atributos que son específicos de ese diseño. Los siguientes son atributos comunes y se aplicarán a todos los diseños:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma exclusiva la vista.
2	android: layout_width Este es el ancho del diseño.
3	android: layout_height Esta es la altura del diseño
4	android: layout_marginTop Este es el espacio adicional en la parte superior del diseño.
5	android: layout_marginBottom Este es el espacio adicional en la parte inferior del diseño.
6	android: layout_marginLeft Este es el espacio adicional en el lado izquierdo del diseño.
7	android: layout_marginRight Este es el espacio adicional en el lado derecho del diseño.
8	android: layout_gravity Esto especifica cómo se colocan las Vistas secundarias.
9	android: layout_weight Esto especifica cuánto espacio adicional en el diseño debe asignarse a la Vista.
10	android: layout_x Esto especifica la coordenada x del diseño.
11	android: layout_y

	Esto especifica la coordenada y del diseño.
12	android: layout_width Este es el ancho del diseño.
13	android: paddingLeft Este es el relleno izquierdo relleno para el diseño.
14	android: paddingRight Este es el relleno correcto para el diseño.
15	android: paddingTop Este es el relleno superior relleno para el diseño.
16	android: paddingBottom Este es el relleno inferior relleno para el diseño.

Aquí el ancho y la altura son la dimensión del diseño / vista que se puede especificar en términos de dp (píxeles independientes de la densidad), sp (píxeles independientes de la escala), pt (puntos que son 1/72 de pulgada), px (Píxeles), mm (Milímetros) y finalmente en (pulgadas).

Puede especificar el ancho y el alto con medidas exactas, pero más a menudo, utilizará una de estas constantes para establecer el ancho o el alto:

- **android: layout_width = wrap_content** le dice a su vista que se **ajuste** a las dimensiones requeridas por su contenido.
- **android: layout_width = fill_parent** le dice a su vista que sea tan grande como su vista principal.

El atributo de gravedad juega un papel importante en el posicionamiento del objeto de vista y puede tomar uno o más (separados por '|') de los siguientes valores constantes.

Constante	Valor	Descripción
top	0x30	Empuje el objeto hacia la parte superior de su contenedor, sin cambiar su tamaño.
bottom	0x50	Empuje el objeto hacia el fondo de su contenedor, sin cambiar su tamaño.
left	0x03	Empuje el objeto hacia la izquierda de su contenedor, sin cambiar su tamaño.

right	0x05	Empuje el objeto hacia la derecha de su contenedor, sin cambiar su tamaño.
center_vertical	0x10	Coloque el objeto en el centro vertical de su contenedor, sin cambiar su tamaño.
fill_vertical	0x70	Aumente el tamaño vertical del objeto si es necesario para que llene completamente su contenedor.
center_horizontal	0x01	Coloque el objeto en el centro horizontal de su contenedor, sin cambiar su tamaño.
fill_horizontal	0x07	Aumente el tamaño horizontal del objeto si es necesario para que llene completamente su contenedor.
centrar	0x11	Coloque el objeto en el centro de su contenedor tanto en el eje vertical como en el horizontal, sin cambiar su tamaño.
fill	0x77	Aumente el tamaño horizontal y vertical del objeto si es necesario para que llene completamente su contenedor.
clip_vertical	0x80	Opción adicional que se puede configurar para que los bordes superior y / o inferior del niño se recorten a los límites de su contenedor. El clip se basará en la gravedad vertical: una gravedad superior recortará el borde inferior, una gravedad inferior recortará el borde superior y ninguna de las dos recortará ambos bordes.
clip_horizontal	0x08	Opción adicional que se puede configurar para que los bordes izquierdo y / o derecho del niño se recorten a los límites de su contenedor. El clip se basará en la gravedad horizontal: una gravedad izquierda recortará el borde derecho, una gravedad derecha recortará el borde izquierdo y ninguna recortará ambos bordes.
start	0x00800003	Empuje el objeto hasta el comienzo de su contenedor, sin cambiar su tamaño.
end	0x00800005	Empuje el objeto hasta el final de su contenedor, sin cambiar su tamaño.

Ver identificación

Un objeto de vista puede tener asignado un ID único que identificará la Vista de manera única dentro del árbol. La sintaxis de un ID dentro de una etiqueta XML es:

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

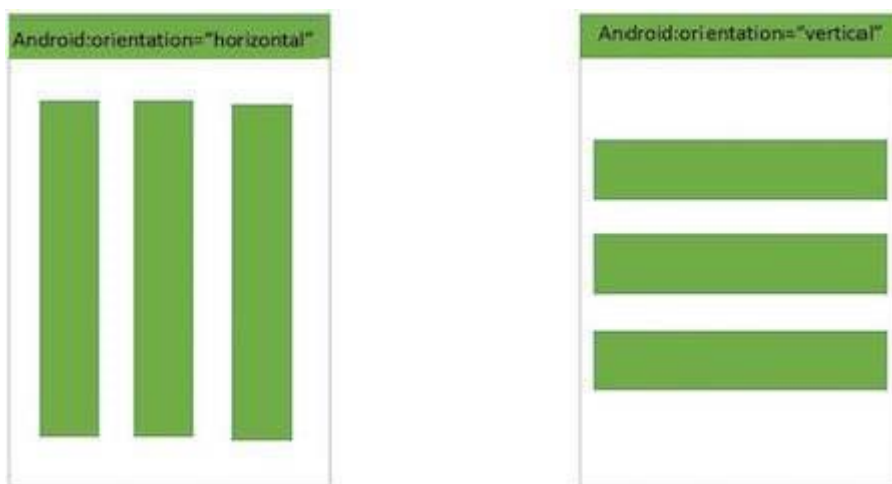
A continuación se muestra una breve descripción de los signos @ y +:

- El símbolo arroba (@) al principio de la cadena indica que el analizador XML debe analizar y expandir el resto de la cadena de identificación e identificarla como un recurso de identificación.
- El símbolo más (+) significa que este es un nuevo nombre de recurso que debe crearse y agregarse a nuestros recursos. Para crear una instancia del objeto de vista y capturarlo del diseño, utilice lo siguiente:

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Diseño lineal de Android

Android LinearLayout es un grupo de vistas que alinea a todos los elementos secundarios de forma *vertical* u *horizontal*.



Disposición lineal

Atributos de LinearLayout

Los siguientes son los atributos importantes específicos de LinearLayout:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.

2	<p>android: baselineAligned</p> <p>Debe ser un valor booleano, ya sea "verdadero" o "falso" y evita que el diseño alinee las líneas base de sus hijos.</p>
3	<p>android: baselineAlignedChildIndex</p> <p>Cuando un diseño lineal forma parte de otro diseño que está alineado con la línea base, puede especificar cuál de sus elementos secundarios se alineará con la línea base.</p>
4	<p>android: divider</p> <p>Se puede dibujar para usar como divisor vertical entre botones. Utiliza un valor de color, en forma de "#rgb", "#argb", "#rrggbb" o "#aarrggbb".</p>
5	<p>android: gravity</p> <p>Esto especifica cómo un objeto debe colocar su contenido, tanto en el eje X como en el Y. Los valores posibles son top, bottom, left, right, center, center_vertical, center_horizontal, etc.</p>
6	<p>android: orientacion</p> <p>Esto especifica la dirección de disposición y utilizará "horizontal" para una fila, "vertical" para una columna. El valor predeterminado es horizontal.</p>
7	<p>android: weightSum</p> <p>Suma del peso del hijo</p>

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando Linear Layout. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla <i>Demo</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir algunos botones en el diseño lineal.

3	No es necesario cambiar las constantes de cadenas, Android Studio se encarga de las cadenas predeterminadas
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

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

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button android:id="@+id/btnStartService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="start_service"/>

    <Button android:id="@+id/btnPauseService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="pause_service"/>

    <Button android:id="@+id/btnStopService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="stop_service"/>

</LinearLayout>
```


A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="action_settings">Settings</string>
</resources>
```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar de la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:

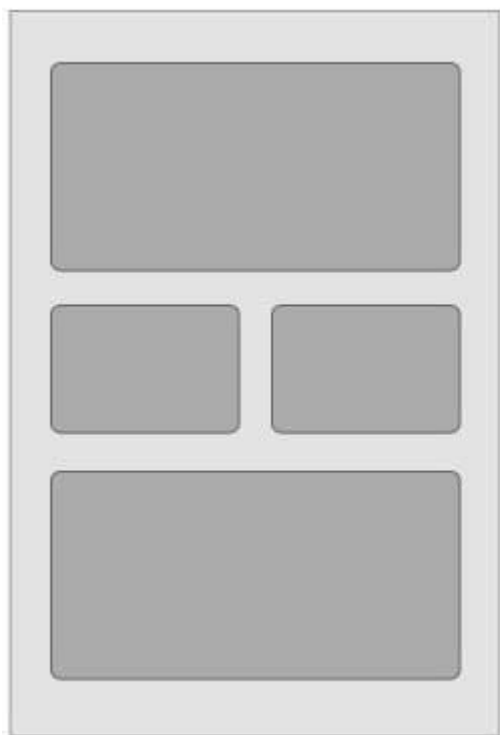


Ahora cambiemos la orientación de Layout como **android: Orientación = "horizontal"** e intentemos ejecutar la misma aplicación, le dará la siguiente pantalla:



Relative Layout

Android RelativeLayout te permite especificar cómo se posicionan las vistas secundarias entre sí. La posición de cada vista se puede especificar como relativa a elementos hermanos o relativa a la principal.



Disposición relativa

Atributos de diseño relativo

Los siguientes son los atributos importantes específicos de RelativeLayout:

No Señor.	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: gravity Esto especifica cómo un objeto debe colocar su contenido, tanto en el eje X como en el Y. Los valores posibles son top, bottom, left, right, center, center_vertical, center_horizontal, etc.
3	android: ignoreGravity Esto indica qué vista no debería verse afectada por la gravedad.

Con RelativeLayout, puede alinear dos elementos por el borde derecho o hacer uno debajo del otro, centrado en la pantalla, centrado a la izquierda, etc. De forma predeterminada, todas las vistas secundarias se dibujan en la parte superior izquierda del diseño, por lo que debe definir la posición de cada vista utilizando las diversas propiedades de diseño disponibles.

en **RelativeLayout.LayoutParams** y algunos de los atributos importantes se indican a continuación:

No Señor.	Atributo y descripción
1	<p>android: layout_above</p> <p>Coloca el borde inferior de esta vista sobre el ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre"</p>
2	<p>android: layout_alignBottom</p> <p>Hace que el borde inferior de esta vista coincida con el borde inferior del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".</p>
3	<p>android: layout_alignLeft</p> <p>Hace que el borde izquierdo de esta vista coincida con el borde izquierdo del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".</p>
4	<p>android: layout_alignParentBottom</p> <p>Si es verdadero, hace que el borde inferior de esta vista coincida con el borde inferior del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".</p>
5	<p>android: layout_alignParentEnd</p> <p>Si es verdadero, hace que el borde final de esta vista coincida con el borde final del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".</p>
6	<p>android: layout_alignParentLeft</p> <p>Si es verdadero, hace que el borde izquierdo de esta vista coincida con el borde izquierdo del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".</p>
7	<p>android: layout_alignParentRight</p> <p>Si es verdadero, hace que el borde derecho de esta vista coincida con el borde derecho del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".</p>
8	<p>android: layout_alignParentStart</p>

	Si es verdadero, hace que el borde inicial de esta vista coincida con el borde inicial del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".
9	android: layout_alignParentTop Si es verdadero, hace que el borde superior de esta vista coincida con el borde superior del padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".
10	android: layout_alignRight Hace que el borde derecho de esta vista coincida con el borde derecho del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
11	android: layout_alignStart Hace que el borde inicial de esta vista coincida con el borde inicial del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
12	android: layout_alignTop Hace que el borde superior de esta vista coincida con el borde superior del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
13	android: layout_below Coloca el borde superior de esta vista debajo del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
14	android: layout_centerHorizontal Si es verdadero, centra este hijo horizontalmente dentro de su padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".
15	android: layout_centerInParent Si es verdadero, centra este hijo horizontal y verticalmente dentro de su padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".
dieciséis	android: layout_centerVertical Si es verdadero, centra este hijo verticalmente dentro de su padre. Debe ser un valor booleano, ya sea "verdadero" o "falso".
17	android: layout_toEndOf



	Coloca el borde de inicio de esta vista al final del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
18	android: layout_toLeftOf Coloca el borde derecho de esta vista a la izquierda del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
19	android: layout_toRightOf Coloca el borde izquierdo de esta vista a la derecha del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".
20	android: layout_toStartOf Coloca el borde final de esta vista al inicio del ID de vista de anclaje dado y debe ser una referencia a otro recurso, en la forma "@ [+] [paquete:] tipo: nombre".

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando Diseño relativo. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World*:

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>demonstración</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir algunos widgets en el diseño relativo.
3	Defina las constantes requeridas en el archivo <i>res / values / strings.xml</i>
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java**. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
```

```
import android.app.Activity;

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

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/name">

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="New Button"
            android:id="@+id/button2" />

    </LinearLayout>

</RelativeLayout>
```


A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="action_settings">Settings</string>
    <string name="reminder">Enter your name</string>
</resources>
```


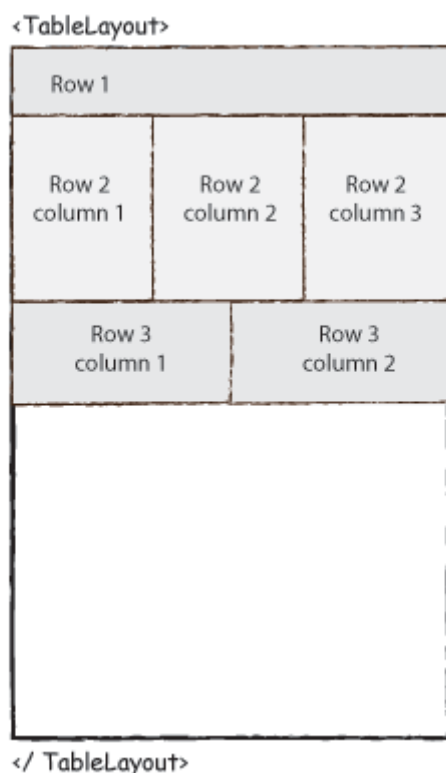
¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:



Table Layout

Android `TableLayout` se organizará en grupos de vistas en filas y columnas. Utilizará el elemento `<TableRow>` para construir una fila en la tabla. Cada fila tiene cero o más celdas; cada celda puede contener un objeto `View`.

Los contenedores `TableLayout` no muestran líneas de borde para sus filas, columnas o celdas.



Atributos de TableLayout

A continuación se muestran los atributos importantes específicos de `TableLayout`:

No Señor.	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: collapseColumns Esto especifica el índice de base cero de las columnas a contraer. Los índices de las columnas deben estar separados por una coma: 1, 2, 5.

3	<p>android: shrinkColumns</p> <p>El índice de base cero de las columnas que se van a reducir. Los índices de las columnas deben estar separados por una coma: 1, 2, 5.</p>
4	<p>android: stretchColumns</p> <p>El índice de base cero de las columnas que se van a estirar. Los índices de las columnas deben estar separados por una coma: 1, 2, 5.</p>

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando Table Layout. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>demonstración</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir algunos widgets en el diseño de la tabla.
3	No es necesario modificar <i>string.xml</i> , Android Studio se encarga de las constantes predeterminadas
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

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

```
}
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :

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

    <TableRow
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView
            android:text="Time"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="1" />

        <TextClock
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/textClock"
            android:layout_column="2" />

    </TableRow>

    <TableRow>

        <TextView
            android:text="First Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="1" />

        <EditText
            android:width="200px"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

    </TableRow>

    <TableRow>

        <TextView
            android:text="Last Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="1" />
```

```
<EditText
    android:width="100px"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</TableRow>

<TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/ratingBar"
        android:layout_column="2" />
</TableRow>

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


<TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

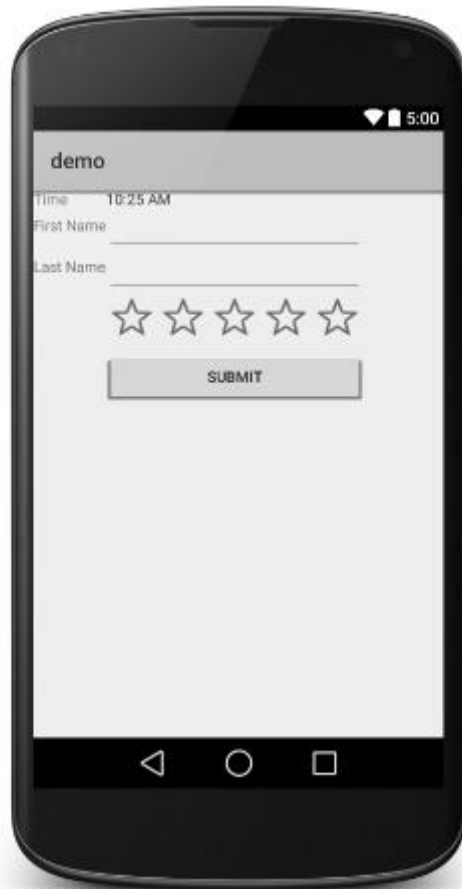
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit"
        android:id="@+id/button"
        android:layout_column="2" />
</TableRow>

</TableLayout>
```

A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="action_settings">Settings</string>
</resources>
```

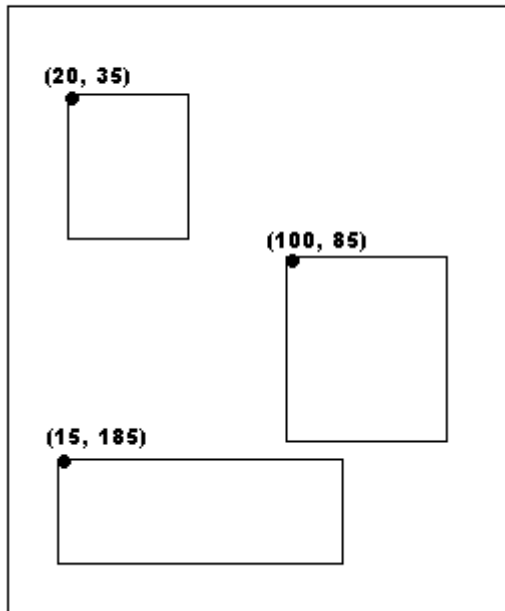
¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:



Absolute Layout

Un diseño absoluto le permite especificar ubicaciones exactas (coordenadas x / y) de sus hijos. Los diseños absolutos son menos flexibles y más difíciles de mantener que otros tipos de diseños sin posicionamiento absoluto.

Absolute Layout



Diseño absoluto

Atributos de AbsoluteLayout

Los siguientes son los atributos importantes específicos de AbsoluteLayout:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: layout_x Esto especifica la coordenada x de la vista.
3	android: layout_y Esto especifica la coordenada y de la vista.

Constructores públicos

AbsoluteLayout (contexto de contexto)

AbsoluteLayout (contexto de contexto, atributos AttributeSet)

AbsoluteLayout (contexto de contexto, atributos AttributeSet, int defStyleAttr)



AbsoluteLayout (contexto de contexto, atributos AttributeSet, int defStyleAttr, int defStyleRes)

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando un diseño absoluto. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>demonstración</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir algunos widgets en diseño absoluto.
3	No es necesario modificar <i>string.xml</i> , Android Studio se encarga de las constantes predeterminadas
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

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

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :


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

```
<Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="OK"
    android:layout_x="50px"
    android:layout_y="361px" />
<Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="Cancel"
    android:layout_x="225px"
    android:layout_y="361px" />

</AbsoluteLayout>
```

A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">demo</string>
    <string name="action_settings">Settings</string>
</resources>
```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:



Frame Layout

El diseño del marco está diseñado para bloquear un área en la pantalla para mostrar un solo elemento. Generalmente, `FrameLayout` debe usarse para contener una sola vista secundaria, porque puede ser difícil organizar las vistas secundarias de una manera que sea escalable a diferentes tamaños de pantalla sin que los elementos secundarios se superpongan entre sí.

Sin embargo, puede agregar varios niños a un `FrameLayout` y controlar su posición dentro del `FrameLayout` asignando gravedad a cada niño, utilizando el atributo android: `layout_gravity`.



Disposición del marco

Atributos de FrameLayout

Los siguientes son los atributos importantes específicos de FrameLayout:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: foreground Esto define el elemento de dibujo para dibujar sobre el contenido y los posibles valores pueden ser un valor de color, en forma de "#rgb", "#argb", "#rrggbb" o "#aarrggbb".
3	android: foregroundGravity

	Define la gravedad que se aplicará al dibujable en primer plano. La gravedad predeterminada se llena. Los valores posibles son top, bottom, left, right, center, center_vertical, center_horizontal, etc.
4	android: measureAllChildren Determina si medir a todos los niños o solo a aquellos en el estado VISIBLE o INVISIBLE al medir. El valor predeterminado es falso.

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando el diseño de marcos. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>demonstración</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir algunos widgets en el diseño del marco.
3	No es necesario cambiar <i>string.xml</i> , Android se encarga de las constantes predeterminadas
4	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;

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

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :


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

    <ImageView
        android:src="@drawable/ic_launcher"
        android:scaleType="fitCenter"
        android:layout_height="250px"
        android:layout_width="250px"/>

    <TextView
        android:text="Frame Demo"
        android:textSize="30px"
        android:textStyle="bold"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:gravity="center"/>
</FrameLayout>
```

A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

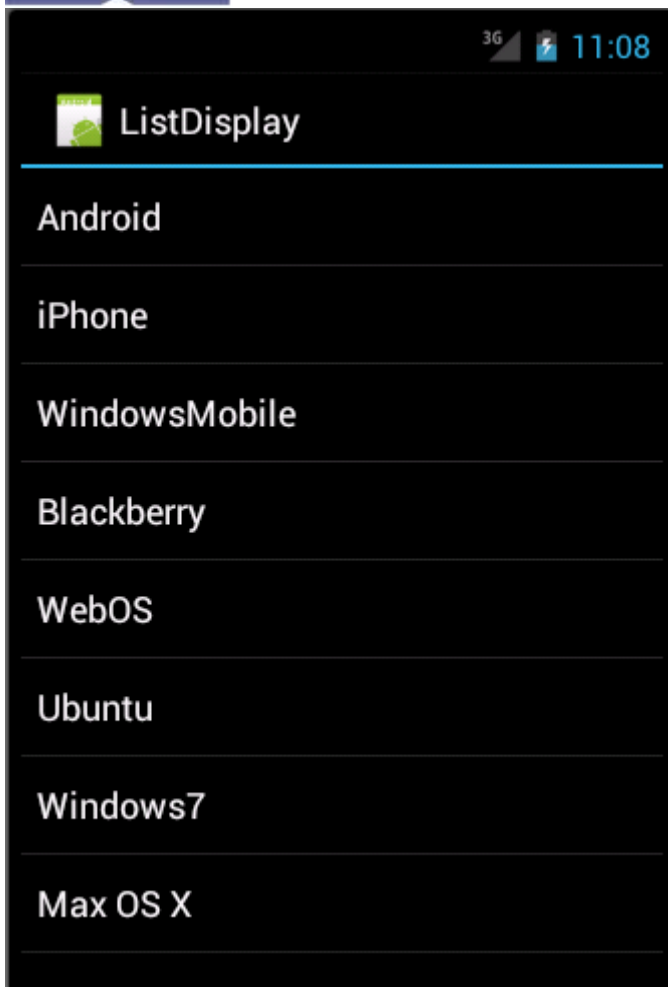
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">demo</string>
    <string name="action_settings">Settings</string>
</resources>
```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:



ListView

Android **ListView** es una vista que agrupa varios elementos y los muestra en una lista de desplazamiento vertical. Los elementos de la lista se insertan automáticamente en la lista mediante un **adaptador** que extrae contenido de una fuente, como una matriz o una base de datos.



Vista de la lista

Un adaptador en realidad crea un puente entre los componentes de la interfaz de usuario y la fuente de datos que completan los datos en el componente de la interfaz de usuario. El adaptador retiene los datos y envía los datos a la vista del adaptador, la vista puede tomar los datos de la vista del adaptador y muestra los datos en diferentes vistas como ruleta, vista de lista, vista de cuadrícula, etc.

El **ListView** y **GridView** son subclases de **AdapterView** y pueden estar ocupados por ellos unirse a un **adaptador**, que recupera datos de una fuente externa y crea una vista que representa cada entrada de datos.

Android proporciona varias subclases de Adapter que son útiles para recuperar diferentes tipos de datos y crear vistas para un AdapterView (es decir, ListView o GridView). Los adaptadores comunes son **ArrayAdapter**, **BaseAdapter**, **CursorAdapter**, **SimpleCursorAdapter**, **SpinnerAdapter** y **WrapperListAdapter**. Veremos ejemplos separados para ambos adaptadores.

Atributos de ListView

Los siguientes son los atributos importantes específicos de GridView:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: divisor Se puede dibujar o colorear para dibujar entre los elementos de la lista.
3	android: dividerHeight Esto especifica la altura del divisor. Esto podría estar en px, dp, sp, in o mm.
4	android: entradas Especifica la referencia a un recurso de matriz que llenará ListView.
5	android: footerDividersEnabled Cuando se establece en falso, ListView no dibujará el divisor antes de cada vista de pie de página. El valor por defecto es verdadero.
6	android: headerDividersEnabled Cuando se establece en falso, ListView no dibujará el divisor después de cada vista de encabezado. El valor por defecto es verdadero.

ArrayAdapter

Puede utilizar este adaptador cuando su fuente de datos sea una matriz. De forma predeterminada, ArrayAdapter crea una vista para cada elemento de la matriz llamando a `toString()` en cada elemento y colocando el contenido en un **TextView**. Considere que tiene una matriz de cadenas que desea mostrar en un ListView, inicie un nuevo **ArrayAdapter** usando un constructor para especificar el diseño de cada cadena y la matriz de cadenas -

```
ArrayAdapter adapter = new  
ArrayAdapter<String>(this, R.layout.ListView, StringArray);
```

Aquí hay argumentos para este constructor:

- El primer argumento **this** es el contexto de aplicación. La mayoría de los casos, mantenerlo **this**.
- El segundo argumento se definirá en el diseño en un archivo XML y tendrá **TextView** para cada cadena en la matriz.

- El argumento final es una matriz de cadenas que se completarán en la vista de texto.

Una vez que haya creado el adaptador de matriz, simplemente llame a **setAdapter ()** en su objeto **ListView** de la siguiente manera:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

Definirá su vista de lista en el directorio `res / layout` en un archivo XML. Para nuestro ejemplo, vamos a utilizar el archivo `activity_main.xml`.

Ejemplo

A continuación se muestra el ejemplo que lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando `ListView`. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World* :

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>ListDisplay</i> en un paquete <i>com.example.ListDisplay</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir contenido <code>ListView</code> con atributos autoexplicativos.
3	No es necesario cambiar <code>string.xml</code> , Android Studio se encarga de las constantes de cadena predeterminadas.
4	Cree un archivo de vista de texto <i>res / layout / activity_listview.xml</i> . Este archivo tendrá la configuración para mostrar todos los elementos de la lista. Para que pueda personalizar sus fuentes, relleno, color, etc. utilizando este archivo.
6	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.ListDisplay / ListDisplay.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.ListDisplay;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;
```

```
public class ListDisplay extends Activity {
    // Array of strings...
    String[] mobileArray =
    {"Android", "IPhone", "WindowsMobile", "Blackberry",
     "WebOS", "Ubuntu", "Windows7", "Max OS X"};

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

        ArrayAdapter adapter = new ArrayAdapter<String>(this,
            R.layout.activity_listview, mobileArray);

        ListView listView = (ListView)
        findViewById(R.id.mobile_list);
        listView.setAdapter(adapter);
    }
}
```

A continuación se muestra el contenido del archivo **res / layout / 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="vertical"
    tools:context=".ListActivity" >

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

</LinearLayout>
```

A continuación se muestra el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

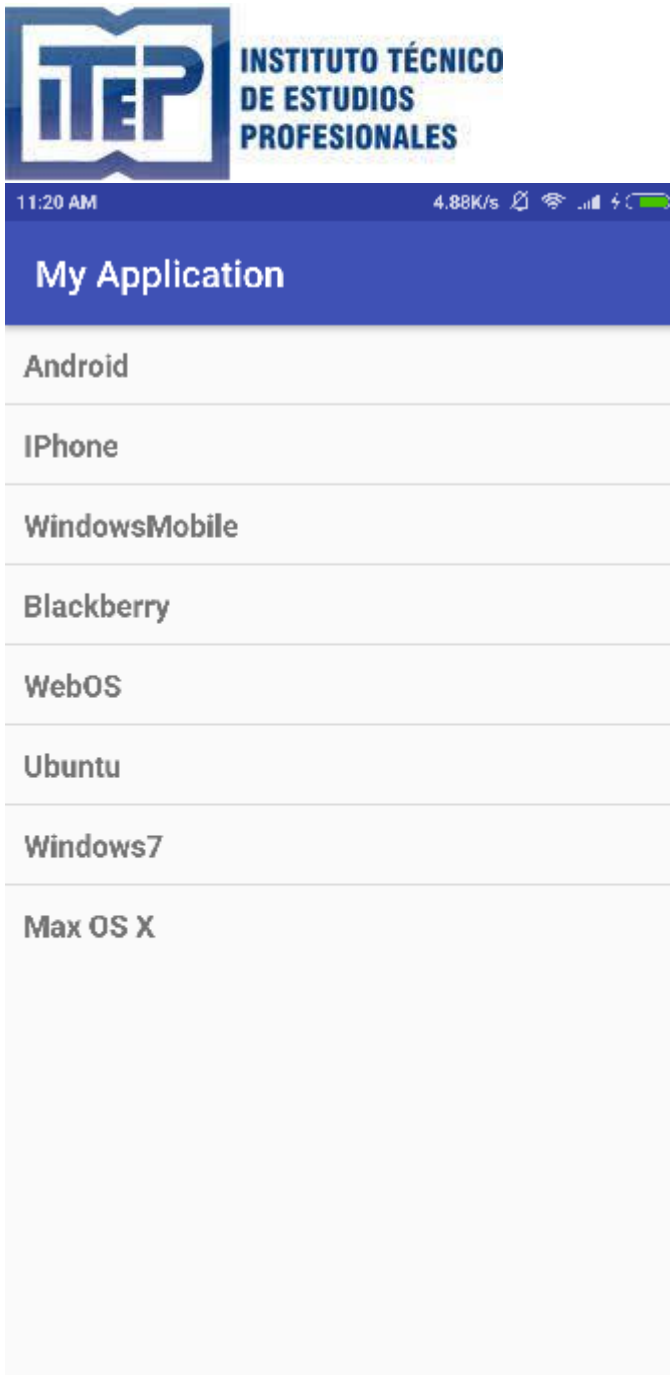
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ListDisplay</string>
    <string name="action_settings">Settings</string>
</resources>
```

A continuación se muestra el contenido del archivo **res / layout / activity_listview.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Single List Item Design -->

<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dip"
    android:textSize="16dip"
    android:textStyle="bold" >
</TextView>
```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que creó su **AVD** mientras configuraba el entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del Emulador:



SimpleCursorAdapter

Puede utilizar este adaptador cuando su fuente de datos sea un Cursor de base de datos. Al usar *SimpleCursorAdapter*, debe especificar un diseño para usar para cada fila en el **Cursor** y qué columnas en el Cursor deben insertarse en qué vistas del diseño.

Por ejemplo, si desea crear una lista de nombres y números de teléfono de personas, puede realizar una consulta que devuelva un Cursor que contiene una fila para cada persona y columnas para los nombres y números. Luego, crea una matriz de cadenas que especifica qué columnas del Cursor desea en el diseño para cada resultado y una matriz de enteros que especifica las vistas correspondientes en las que se debe colocar cada columna:

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
    ContactsContract.CommonDataKinds.Phone.NUMBER};
```

```
int[] toViews = {R.id.display_name, R.id.phone_number};
```

Cuando crea una instancia de SimpleCursorAdapter, pase el diseño que se usará para cada resultado, el Cursor que contiene los resultados y estas dos matrices:

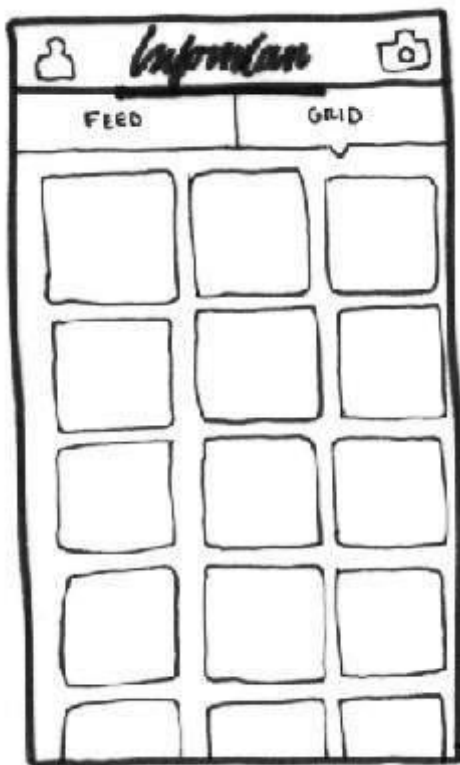
```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
    R.layout.person_name_and_number, cursor, fromColumns,
    toViews, 0);

ListView listView = getListView();
listView.setAdapter(adapter);
```

El SimpleCursorAdapter luego crea una vista para cada fila en el Cursor usando el diseño proporcionado insertando cada elemento de Columns en la vista **toViews** correspondiente.

Grid View

Android **GridView** muestra elementos en una cuadrícula de desplazamiento bidimensional (filas y columnas) y los elementos de la cuadrícula no están necesariamente predeterminados, pero se insertan automáticamente en el diseño mediante un **ListAdapter**



Vista en cuadrícula

Un adaptador en realidad crea un puente entre los componentes de la interfaz de usuario y la fuente de datos que completan los datos en el componente de la interfaz de usuario. El adaptador se puede utilizar para suministrar datos como ruleta, vista de lista, vista de cuadrícula, etc.

El **ListView** y **GridView** son subclases de **AdapterView** y pueden estar ocupados por ellos unirse a un **adaptador**, que recupera datos de una fuente externa y crea una vista que representa cada entrada de datos.

Atributos de GridView

Los siguientes son los atributos importantes específicos de GridView:

No Señor	Atributo y descripción
1	android: id Este es el ID que identifica de forma única el diseño.
2	android: columnWidth Esto especifica el ancho fijo para cada columna. Esto podría estar en px, dp, sp, in o mm.
3	android: gravity Especifica la gravedad dentro de cada celda. Los valores posibles son top, bottom, left, right, center, center_vertical, center_horizontal, etc.
4	android: horizontalSpacing Define el espaciado horizontal predeterminado entre columnas. Esto podría estar en px, dp, sp, in o mm.
5	android: numColumns Define cuántas columnas mostrar. Puede ser un valor entero, como "100" o auto_fit, lo que significa mostrar tantas columnas como sea posible para llenar el espacio disponible.
6	android: stretchMode Define cómo se deben estirar las columnas para llenar el espacio vacío disponible, si lo hay. Este debe ser cualquiera de los valores: <ul style="list-style-type: none"> • none: el estiramiento está desactivado. • spacingWidth: el espacio entre cada columna se estira. • columnWidth: cada columna se estira por igual.

	<ul style="list-style-type: none"> spacingWidthUniform: el espacio entre cada columna se estira uniformemente.
7	<p>android: verticalSpacing</p> <p>Define el espaciado vertical predeterminado entre filas. Esto podría estar en px, dp, sp, in o mm.</p>

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando GridView. Siga los siguientes pasos para modificar la aplicación de Android que creamos en el capítulo *Ejemplo de Hello World*:

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>HelloWorld</i> en un paquete <i>com.example.helloworld</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el contenido detallado del archivo <i>res / layout / activity_main.xml</i> para incluir contenido GridView con atributos autoexplicativos.
3	No es necesario cambiar <i>string.xml</i> , Android Studio se encarga de las cadenas predeterminadas que se colocan en <i>string.xml</i>
4	Pongamos algunas imágenes en la carpeta <i>res / drawable-hdpi</i> . He puesto <i>sample0.jpg</i> , <i>sample1.jpg</i> , <i>sample2.jpg</i> , <i>sample3.jpg</i> , <i>sample4.jpg</i> , <i>sample5.jpg</i> , <i>sample6.jpg</i> y <i>sample7.jpg</i> .
5	Cree una nueva clase llamada ImageAdapter en un paquete <i>com.example.helloworld</i> que amplíe <i>BaseAdapter</i> . Esta clase implementará la funcionalidad de un adaptador que se utilizará para llenar la vista.
6	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.helloworld / MainActivity.java**. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.GridView;
```

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

        GridView gridview = (GridView)
findViewById(R.id.gridview);
        gridview.setAdapter(new ImageAdapter(this));
    }
}
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<GridView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

A continuación se muestra el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="action_settings">Settings</string>
</resources>
```

A continuación se muestra el contenido del archivo **src / com.example.helloworld / ImageAdapter.java** :

```
package com.example.helloworld;

import android.content.Context;

import android.view.View;
import android.view.ViewGroup;

import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
```

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    // Constructor
    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the
    Adapter
    public View getView(int position, View convertView,
        ViewGroup parent) {
        ImageView imageView;

        if (convertView == null) {
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new
GridView.LayoutParams(85, 85));
imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        }
        else
        {
            imageView = (ImageView) convertView;
        }
        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }


    // Keep all Images in array
    public Integer[] mThumbIds = {
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
    }
}
```

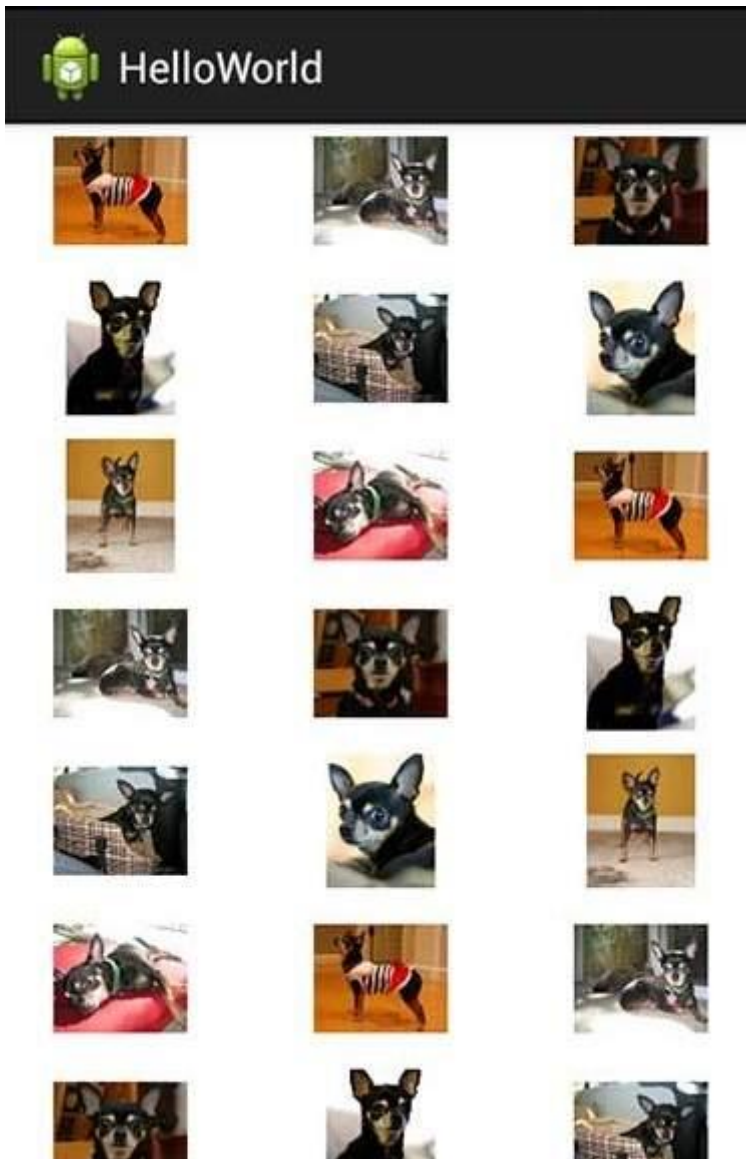


```

        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7
    };
}

```

¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar en la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:



Ejemplo de subactividad

Extendamos la funcionalidad del ejemplo anterior donde mostraremos la imagen de cuadrícula seleccionada en pantalla completa. Para lograr esto, necesitamos introducir una nueva actividad. Solo tenga en cuenta que para cualquier actividad, necesitamos realizar todos los pasos como para implementar una clase de actividad, definir esa actividad en el archivo `AndroidManifest.xml`, definir el diseño relacionado y finalmente vincular esa subactividad con la actividad principal en el archivo principal. clase de actividad. Así que sigamos los pasos para modificar el ejemplo anterior:

Paso	Descripción
1	Utilizará Android Studio IDE para crear una aplicación de Android y nombrarla como <i>HelloWorld</i> en un paquete <i>com.example.helloworld</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Cree una nueva clase de actividad como <i>SingleViewActivity.java</i> en un paquete <i>com.example.helloworld</i> como se muestra a continuación.
3	Cree un nuevo archivo de diseño para la nueva actividad en la carpeta res / layout / . Llamemos a este archivo XML como <i>single_view.xml</i> .
4	Defina su nueva actividad en el archivo <i>AndroidManifest.xml</i> usando la etiqueta <code><activity ... /></code> . Una aplicación puede tener una o más actividades sin restricciones.
5	Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación.

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.helloworld / MainActivity.java**. Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.helloworld;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

import android.view.Menu;
import android.view.View;

import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;

public class MainActivity extends Activity {
    @Override
```

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

    GridView gridview = (GridView)
    findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new
    OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent,
        View v, int position, long id){
            // Send intent to SingleViewActivity
            Intent i = new Intent(getApplicationContext(),
            SingleViewActivity.class);
            // Pass image index
            i.putExtra("id", position);
            startActivity(i);
        }
    });
}
```

A continuación se **muestra** el contenido del nuevo archivo de actividad **src / com.example.helloworld / SingleViewActivity.java** file -

```
package com.example.helloworld;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.ImageView;

public class SingleViewActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.single_view);

        // Get intent data
        Intent i = getIntent();

        // Selected image id
        int position = i.getExtras().getInt("id");
        ImageAdapter imageAdapter = new ImageAdapter(this);

        ImageView imageView = (ImageView)
        findViewById(R.id.SingleView);

        imageView.setImageResource(imageAdapter.mThumbIds[position]);
    }
}
```



```
}
```

A continuación se muestra el contenido del archivo **res / layout / single_view.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" >

    <ImageView android:id="@+id/SingleView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</LinearLayout>
```

A continuación se **muestra** el contenido de **AndroidManifest.xml** para definir dos nuevas constantes:

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.helloworld.MainActivity"
            android:label="@string/app_name" >

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

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

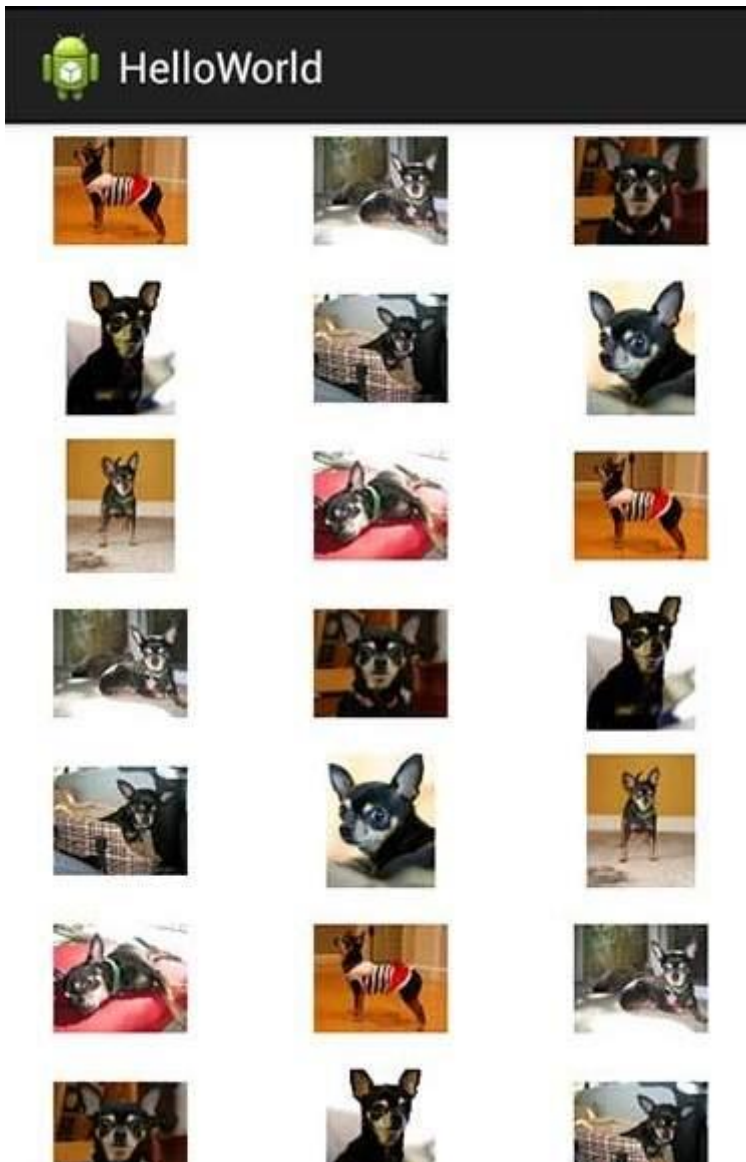
        </activity>

        <activity
            android:name=".SingleViewActivity"></activity>

    </application>
</manifest>
```



¡Intentemos ejecutar nuestro **Hello World** modificado ! aplicación que acabamos de modificar. Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio abra uno de los archivos de actividad de su proyecto y haga clic en el icono Ejecutar de la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:



Ahora, si hace clic en cualquiera de las imágenes, se mostrará como una sola imagen, por ejemplo



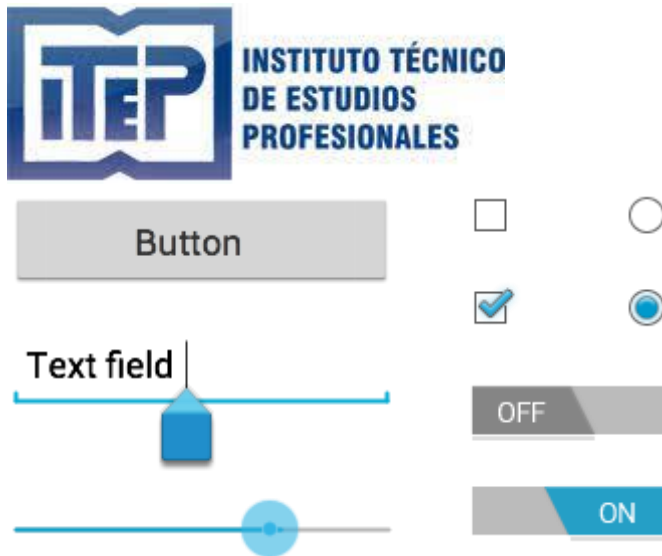
HelloWorld



Tenga en cuenta que las imágenes mencionadas anteriormente se han tomado del sitio web oficial de Android.

Controles de la interfaz de usuario

Los controles de entrada son los componentes interactivos de la interfaz de usuario de su aplicación. Android proporciona una amplia variedad de controles que puede usar en su interfaz de usuario, como botones, campos de texto, barras de búsqueda, casillas de verificación, botones de zoom, botones de alternancia y muchos más.



Elementos de la interfaz de usuario

Una **vista** es un objeto que dibuja algo en la pantalla con el que el usuario puede interactuar y un **ViewGroup** es un objeto que contiene otros objetos View (y ViewGroup) para definir el diseño de la interfaz de usuario.

Usted define su diseño en un archivo XML que ofrece una estructura legible por humanos para el diseño, similar a HTML. Por ejemplo, un diseño vertical simple con una vista de texto y un botón se ve así:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />

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

Controles de IU de Android

Hay una serie de controles de IU proporcionados por Android que le permiten crear la interfaz gráfica de usuario para su aplicación.

Control y descripción de la interfaz de usuario	
No Señor.	
1	<p>TextView</p> <p>Este control se utiliza para mostrar texto al usuario.</p>

2	<p>EditText</p> <p>EditText es una subclase predefinida de TextView que incluye amplias capacidades de edición.</p>
3	<p>AutoCompleteTextView</p> <p>AutoCompleteTextView es una vista similar a EditText, excepto que muestra una lista de sugerencias de finalización automáticamente mientras el usuario escribe.</p>
4	<p>Button</p> <p>Un botón que el usuario puede presionar o hacer clic para realizar una acción.</p>
5	<p>ImageButton</p> <p>Un ImageButton es un AbsoluteLayout que le permite especificar la ubicación exacta de sus hijos. Esto muestra un botón con una imagen (en lugar de texto) que el usuario puede presionar o hacer clic en él.</p>
6	<p>CheckBox</p> <p>Un interruptor de encendido / apagado que el usuario puede activar. Debe utilizar la casilla de verificación al presentar a los usuarios un grupo de opciones seleccionables que no se excluyen mutuamente.</p>
7	<p>ToggleButton</p> <p>Un botón de encendido / apagado con indicador de luz.</p>
8	<p>RadioButton</p> <p>El RadioButton tiene dos estados: marcado o no marcado.</p>
9	<p>RadioGroup</p> <p>Un RadioGroup se utiliza para agrupar uno o más RadioButtons.</p>
10	<p>ProgressBar</p> <p>La vista ProgressBar proporciona comentarios visuales sobre algunas tareas en curso, como cuando está realizando una tarea en segundo plano.</p>
11	<p>Spinner</p> <p>Una lista desplegable que permite a los usuarios seleccionar un valor de un conjunto.</p>
12	<p>TimePicker</p> <p>La vista TimePicker permite a los usuarios seleccionar una hora del día, ya sea en el modo de 24 horas o en el modo AM / PM.</p>
13	<p>DatePicker</p>

La vista DatePicker permite a los usuarios seleccionar una fecha del día.

Crear controles de IU

Los controles de entrada son los componentes interactivos de la interfaz de usuario de su aplicación. Android proporciona una amplia variedad de controles que puede usar en su interfaz de usuario, como botones, campos de texto, barras de búsqueda, casillas de verificación, botones de zoom, botones de alternancia y muchos más.

Como se explicó en el capítulo anterior, un objeto de vista puede tener asignado un ID único que identificará la Vista de manera única dentro del árbol. La sintaxis de un ID dentro de una etiqueta XML es:

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

Para crear un control / vista / widget de la interfaz de usuario, deberá definir una vista / widget en el archivo de diseño y asignarle una ID única de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/text_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
</LinearLayout>
```

Luego, finalmente cree una instancia del objeto Control y captúrelo del diseño, use lo siguiente:

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Control TextView

Un **TextView** muestra texto al usuario y, opcionalmente, le permite editarlo. Un TextView es un editor de texto completo, sin embargo, la clase básica está configurada para no permitir la edición.

Atributos de TextView

A continuación se muestran los atributos importantes relacionados con el control TextView. Puede consultar la documentación oficial de Android para obtener una lista completa de atributos y métodos relacionados que puede usar para cambiar estos atributos en tiempo de ejecución.

No Señor.	Atributo y descripción
1	<p>android: id</p> <p>Este es el ID que identifica de forma única al control.</p>
2	<p>android: capitalice</p> <p>Si se establece, especifica que este TextView tiene un método de entrada textual y debe escribir en mayúscula automáticamente lo que escribe el usuario.</p> <ul style="list-style-type: none"> • No escriba nada en mayúsculas automáticamente - 0 • Escriba con mayúscula la primera palabra de cada oración - 1 • Ponga en mayúscula la primera letra de cada palabra - 2 • Capitalice cada carácter - 3
3	<p>android: cursorVisible</p> <p>Hace que el cursor sea visible (predeterminado) o invisible. El valor predeterminado es falso.</p>
4	<p>android: editable</p> <p>Si se establece en verdadero, especifica que este TextView tiene un método de entrada.</p>
5	<p>android: fontFamily</p> <p>Familia de fuentes (nombrada por cadena) para el texto.</p>
6	<p>android: gravity</p> <p>Especifica cómo alinear el texto con los ejes xy / o y de la vista cuando el texto es más pequeño que la vista.</p>
7	<p>android: hint</p> <p>Sugerencia de texto para mostrar cuando el texto está vacío.</p>
8	<p>android: inputType</p> <p>El tipo de datos que se colocan en un campo de texto. Teléfono, fecha, hora, número, contraseña, etc.</p>
9	<p>Android: maxHeight</p> <p>Hace que TextView tenga como máximo esta cantidad de píxeles de alto.</p>

10	<p>android: maxWidth</p> <p>Hace que TextView tenga como máximo esta cantidad de píxeles de ancho.</p>
11	<p>Android: minHeight</p> <p>Hace que TextView tenga al menos esta cantidad de píxeles de alto.</p>
12	<p>android: minWidth</p> <p>Hace que TextView tenga al menos esta cantidad de píxeles de ancho.</p>
13	<p>android: password</p> <p>Si los caracteres del campo se muestran como puntos de contraseña en lugar de ellos mismos. Valor posible "verdadero" o "falso".</p>
14	<p>android: phoneNumber</p> <p>Si se establece, especifica que este TextView tiene un método de entrada de número de teléfono. Valor posible "verdadero" o "falso".</p>
15	<p>android: text</p> <p>Texto para mostrar.</p>
dieciséis	<p>android: textAllCaps</p> <p>Presente el texto en MAYÚSCULAS. Valor posible "verdadero" o "falso".</p>
17	<p>android: textColor</p> <p>Color de texto. Puede ser un valor de color, en forma de "#rgb", "#argb", "#rrggbb" o "#aarrggbb".</p>
18	<p>android: textColorHighlight</p> <p>Color del resaltado de la selección de texto.</p>
19	<p>android: textColorHint</p> <p>Color del texto de la sugerencia. Puede ser un valor de color, en forma de "#rgb", "#argb", "#rrggbb" o "#aarrggbb".</p>
20	<p>android: textIsSelectable</p>

	Indica que se puede seleccionar el contenido de un texto no editable. Valor posible "verdadero" o "falso".
21	<p>android: textSize</p> <p>Tamaño del texto. El tipo de dimensión recomendado para el texto es "sp" para píxeles escalados (ejemplo: 15 sp).</p>
22	<p>android: textStyle</p> <p>Estilo (negrita, cursiva, negrita e itálica) para el texto. Puede utilizar o más de los siguientes valores separados por ' '. <ul style="list-style-type: none"> • normal - 0 • negrita - 1 • cursiva - 2 </p>
23	<p>android: typeface</p> <p>Tipo de letra (normal, sans, serif, monoespacio) para el texto. Puede utilizar o más de los siguientes valores separados por ' '. <ul style="list-style-type: none"> • normal - 0 • sans - 1 • serif - 2 • monoespacio - 3 </p>

Ejemplo

Este ejemplo lo llevará a través de sencillos pasos para mostrar cómo crear su propia aplicación de Android usando Linear Layout y TextView.

Paso	Descripción
1	Utilizará Android Studio para crear una aplicación de Android y nombrarla como <i>demostración</i> en un paquete <i>com.example.demo</i> como se explica en el capítulo <i>Ejemplo de Hello World</i> .
2	Modifique el archivo <i>src / MainActivity.java</i> para agregar el código necesario.
2	Modifique el contenido predeterminado del archivo <i>res / layout / activity_main.xml</i> para incluir el control de la interfaz de usuario de Android.
3	No es necesario cambiar las constantes de cadena predeterminadas en el archivo <i>string.xml</i> . Android Studio se encarga de las constantes de cadena predeterminadas.

- | | |
|----------|--|
| 4 | Ejecute la aplicación para iniciar el emulador de Android y verifique el resultado de los cambios realizados en la aplicación. |
|----------|--|

A continuación se muestra el contenido del archivo de actividad principal modificado **src / com.example.demo / MainActivity.java** . Este archivo puede incluir cada uno de los métodos fundamentales del ciclo de vida.

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

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

        //--- text view---
        TextView txtView = (TextView)
        findViewById(R.id.text_id);
    }
}
```

A continuación se muestra el contenido del archivo **res / layout / activity_main.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=".MainActivity" >

    <TextView
        android:id="@+id/text_id"
        android:layout_width="300dp"
        android:layout_height="200dp"
        android:capitalize="characters"
        android:text="hello_world"
        android:textColor="@android:color/holo_blue_dark"
        android:textColorHighlight="@android:color/primary_text_dark"
```

```
android:layout_centerVertical="true"  
android:layout_alignParentEnd="true"  
android:textSize="50dp"/>
```


```
</RelativeLayout>
```

A continuación se **muestra** el contenido de **res / values / strings.xml** para definir dos nuevas constantes:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">demo</string>  
</resources>
```

A continuación se muestra el contenido predeterminado de **AndroidManifest.xml** :

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest  
    xmlns:android="http://schemas.android.com/apk/res/android"  
        package="com.example.demo" >  
  
    <application  
        android:allowBackup="true"  
        android:icon="@drawable/ic_launcher"  
        android:label="@string/app_name"  
        android:supportsRtl="true"  
        android:theme="@style/AppTheme" >  
  
        <activity  
            android:name="com.example.demo.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>  
  
    </application>  
</manifest>
```

Intentemos ejecutar su aplicación de **demonstración** . Supongo que ha creado su **AVD** mientras realizaba la configuración del entorno. Para ejecutar la aplicación desde Android Studio, abra uno de los archivos de actividad de su proyecto y haga clic en el  icono Ejecutar de la barra de herramientas. Android Studio instala la aplicación en su AVD y la inicia y, si todo está bien con su configuración y aplicación, se mostrará la siguiente ventana del emulador:



Ejercicio

Recomendaré probar el ejemplo anterior con diferentes atributos de `TextView` en el archivo XML de diseño, así como en el momento de la programación para tener un aspecto diferente del `TextView`. Intente hacerlo editable, cambie el color de la fuente, la familia de la fuente, el ancho, el tamaño del texto, etc. y vea el resultado. También puede probar el ejemplo anterior con múltiples controles `TextView` en una actividad.