Clase 3

Curso Desarrollo en ANDROID V2.0

Emiliano Gonzalez

Pedro Coronel

os los Derechos Reservados- Emiliano Gonzalez/Pedro Coronel

Asunción – Paraguay

2013

ANDROID
DEVELOPER

Controles Básicos

En los próximos apartados vamos a hacer un repaso de los diferentes controles que pone a nuestra disposición la plataforma de desarrollo de este sistema operativo. Empezaremos con los controles más básicos y seguiremos posteriormente con algunos algo más elaborados.

En esta primera parada sobre el tema nos vamos a centrar en los diferentes tipos de botones y cómo podemos personalizarlos. El SDK de Android nos proporciona tres tipos de botones: el clásico (Button), el de tipo on/off (ToggleButton), y el que puede contener una imagen (ImageButton). En la imagen siguiente vemos el aspecto por defecto de estos tres controles.

No vamos a comentar mucho sobre ellos dado que son controles de sobra conocidos por todos, ni vamos a enumerar todas sus propiedades porque existen decenas. A modo de referencia, a medida que los vayamos comentando iré poniendo enlaces a su página de la documentación oficial de Android para poder consultar todas sus propiedades en caso de necesidad.

Control Button [API]

Un control de tipo Button es el botón más básico que podemos utilizar. En el ejemplo siguiente definimos un botón con el texto "Púlsame" asignando su propiedad android:text. Además de esta propiedad podríamos utilizar muchas otras como el color de fondo (android:background), estilo de fuente (android:typeface), color de fuente (android:textcolor), tamaño de fuente (android:textSize), etc.

```
<Button android:id="@+id/BtnBoton1"
    android:text="Púlsame"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Control ToggleButton [API]

Un control de tipo ToggleButton es un tipo de botón que puede permanecer en dos estados, pulsado/no_pulsado. En este caso, en vez de definir un sólo texto para el control definiremos dos, dependiendo de su estado. Así, podremos asignar las propiedades android:textOn y android:textoOff para definir ambos textos.

Veamos un ejemplo a continuación.

```
<ToggleButton android:id="@+id/BtnBoton2"
    android:textOn="ON"
    android:textOff="OFF"
```

Curso Android V2- 2012 -Todos los Derechos Reservados Emilio Gonzalez/Pedro Cordoniel

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

Control ImageButton [API]

En un control de tipo ImageButton podremos definir una imagen a mostrar en vez de un texto, para lo que deberemos asignar la propiedad android:src. Normalmente asignaremos esta propiedad con el descriptor de algún recurso que hayamos incluido en la carpeta /res/drawable. Así, por ejemplo, en nuestro caso hemos incluido una imagen llamada "ok.png" por lo que haremos referencia al recurso "@drawable/ok".

```
<ImageButton android:id="@+id/BtnBoton3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ok" />
```

Eventos de un botón

Como podéis imaginar, aunque estos controles pueden lanzar muchos otros eventos, el más común de todos ellos y el que querremos capturar en la mayoría de las ocasiones es el evento onClick. Para definir la lógica de este evento tendremos que implementarla definiendo un nuevo objeto View.OnClickListener() y asociándolo al botón mediante el método setOnClickListener(). La forma más habitual de hacer esto es la siguiente:

```
final Button btnBoton1 = (Button)findViewById(R.id.BtnBoton1);
btnBoton1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0)
    {
        lblMensaje.setText("Botón 1 pulsado!");
    }
});
```

En el caso de un botón de tipo ToggleButton suele ser de utilidad conocer en qué estado ha quedado el botón tras ser pulsado, para lo que podemos utilizar su método isChecked(). En el siguiente ejemplo se comprueba el estado del botón tras ser pulsado y se realizan acciones distintas según el resultado.

```
final ToggleButton btnBoton2 =
(ToggleButton)findViewById(R.id.BtnBoton2);
btnBoton2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0)
    {
        if(btnBoton2.isChecked())
            lblMensaje.setText("Botón 2: ON");
        else
            lblMensaje.setText("Botón 2: OFF");
```

Curso Android V2 - 2013 - Todos los Derechos Reservados Ernilmo Gonzalez/Pedro Coronel

```
    }
});
```

Personalizar el aspecto un botón [y otros controles]

En la imagen anterior vimos el aspecto que presentan por defecto los tres tipos de botones disponibles. Pero, ¿y si quisiéramos personalizar su aspecto más allá de cambiar un poco el tipo o el color de la letra o el fondo?

Para cambiar la forma de un botón podríamos simplemente asignar una imagen a la propiedad android:background, pero esta solución no nos serviría de mucho porque siempre se mostraría la misma imagen incluso con el botón pulsado, dando poca sensación de elemento "clickable".

La solución perfecta pasaría por tanto por definir diferentes imágenes de fondo dependiendo del estado del botón. Pues bien, Android nos da total libertad para hacer esto mediante el uso de selectores. Un selector se define mediante un fichero XML localizado en la carpeta /res/drawable, y en él se pueden establecer los diferentes valores de una propiedad determinada de un control dependiendo de su estado.

Por ejemplo, si quisiéramos dar un aspecto plano a un botón ToggleButton, podríamos diseñar las imágenes necesarias para los estados "pulsado" (en el ejemplo toggle_on.png) y "no pulsado" (en el ejemplo toggle_off.png) y crear un selector como el siguiente:

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

    <item android:state_checked="false"
            android:drawable="@drawable/toggle_off" />
    <item android:state_checked="true"
            android:drawable="@drawable/toggle_on" />

</selector>
```

Este selector lo guardamos por ejemplo en un fichero llamado toggle_style.xml y lo colocamos como un recurso más en nuestra carpeta de recursos /res/drawable. Hecho esto, tan sólo bastaría hacer referencia a este nuevo recurso que hemos creado en la propiedad android:background del botón:

```
 <ToggleButton android:id="@+id/BtnBoton4"
    android:textOn="ON"
    android:textOff="OFF"
    android:padding="10dip"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/toggle_style"/>
```

Curso Android V2 - 2013 Todos los Derechos Reservados Emiliano Gonzalez/Pedro Corbinel

En la siguiente imagen vemos el aspecto por defecto de un ToggleButton y cómo ha quedado nuestro ToggleButton personalizado.

Imágenes, etiquetas y cuadros de texto

En este apartado nos vamos a centrar en otros tres componentes básicos imprescindibles en nuestras aplicaciones: las imágenes (ImageView), las etiquetas (TextView) y por último los cuadros de texto (EditText).

Control ImageView [API]

El control ImageView permite mostrar imágenes en la aplicación. La propiedad más interesante es android:src, que permite indicar la imagen a mostrar. Nuevamente, lo normal será indicar como origen de la imagen el identificador de un recurso de nuestra carpeta /res/drawable, por ejemplo android:src="@drawable/unaimagen".

Además de esta propiedad, existen algunas otras útiles en algunas ocasiones como las destinadas a establecer el tamaño máximo que puede ocupar la imagen, android:maxWidth y android:maxHeight.

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

En la lógica de la aplicación, podríamos establecer la imagen mediante el método setImageResorce(…), pasándole el ID del recurso a utilizar como contenido de la imagen.

```
ImageView img = (ImageView)findViewById(R.id.ImgFoto);
img.setImageResource(R.drawable.icon);
```

Control TextView [API]

El control TextView es otro de los clásicos en la programación de GUIs, las etiquetas de texto, y se utiliza para mostrar un determinado texto al usuario. Al igual que en el caso de los botones, el texto del control se establece mediante la propiedad android:text. A parte de esta propiedad, la naturaleza del control hace que las más interesantes sean las que establecen el formato del texto mostrado, que al igual que en el caso de los botones son las siguientes: android:background (color de fondo), android:textColor (color del texto), android:textSize (tamaño de la fuente) y android:typeface (estilo del texto: negrita, cursiva, …).

```
<TextView android:id="@+id/LblEtiqueta"
android:layout_width="fill_parent"
```

Curso Android V2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

```
android:layout_height="wrap_content"
android:text="Escribe algo:"
android:background="#AA44FF"
android:typeface="monospace" />
```

De igual forma, también podemos manipular estas propiedades desde nuestro código. Como ejemplo, en el siguiente fragmento recuperamos el texto de una etiqueta con getText(), y posteriormente le concatenamos unos números, actualizamos su contenido mediante setText() y le cambiamos su color de fondo con setBackgroundColor().

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);
String texto = lblEtiqueta.getText().toString();
texto += "123";
lblEtiqueta.setText(texto);
```

Control EditText [API]

El control EditText es el componente de edición de texto que proporciona la plataforma Android. Permite la introducción y edición de texto por parte del usuario, por lo que en tiempo de diseño la propiedad más interesante a establecer, además de su posición/tamaño y formato, es el texto a mostrar, atributo android:text.

```
<EditText android:id="@+id/TxtTexto"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_below="@id/LblEtiqueta" />
```

De igual forma, desde nuestro código podremos recuperar y establecer este texto mediante los métodos getText() y setText(nuevoTexto) respectivamente:

```
final EditText txtTexto = (EditText)findViewById(R.id.TxtTexto);
String texto = txtTexto.getText().toString();
txtTexto.setText("Hola mundo!");
```

Un detalle que puede haber pasado desapercibido. ¿Os habéis fijado en que hemos tenido que hacer un toString() sobre el resultado de getText()? La explicación para esto es que el método getText() no devuelve un String sino un objeto de tipo Editable, que a su vez implementa la interfaz Spannable. Y esto nos lleva a la característica más interesante el control EditText, y es que no sólo nos permite editar texto plano sino también texto enriquecido o con formato. Veamos cómo y qué opciones tenemos, y para empezar comentemos algunas cosas sobre los objetos Spannable.

Interfaz Spanned

Un objeto de tipo Spanned es algo así como una cadena de caracteres (deriva de la interfaz CharSequence) en la que podemos insertar otros objetos a modo de marcas o etiquetas (spans) asociados a rangos de caracteres. De esta interfaz deriva la interfaz Spannable, que permite la

Curso Android v2 2013 - Todos los Derechos Reservados - Enrique Gonzalez/Pedro Coronel

modificación de estas marcas, y a su vez de ésta última deriva la interfaz Editable, que permite además la modificación del texto.

Aunque en el apartado en el que nos encontramos nos interesaremos principalmente por las marcas de formato de texto, en principio podríamos insertar cualquier tipo de objeto.

Existen muchos tipos de spans predefinidos en la plataforma que podemos utilizar para dar formato al texto, entre ellos:

- TypefaceSpan. Modifica el tipo de fuente.
- StyleSpan. Modifica el estilo del texto (negrita, cursiva, …).
- ForegroudColorSpan. Modifica el color del texto.
- AbsoluteSizeSpan. Modifica el tamaño de fuente.

De esta forma, para crear un nuevo objeto Editable e insertar una marca de formato podríamos hacer lo siguiente:

```
//Creamos un nuevo objeto de tipo Editable
Editable str = Editable.Factory.getInstance().newEditable("Esto es un
simulacro.");

//Marcamos cono fuente negrita la palabra "simulacro"
str.setSpan(new StyleSpan(android.graphics.Typeface.BOLD), 11, 19,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

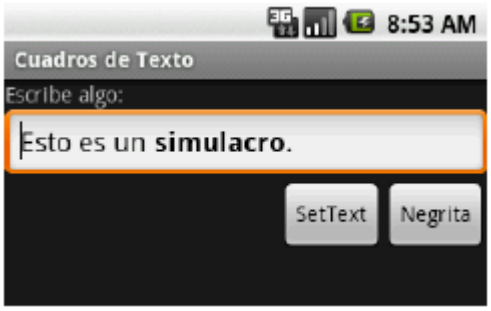
En este ejemplo estamos insertando un span de tipo StyleSpan para marcar un fragmento de texto con estilo negrita. Para insertarlo utilizamos el método setSpan(), que recibe como parámetro el objeto Span a insertar, la posición inicial y final del texto a marcar, y un flag que indica la forma en la que el span se podrá extender al insertarse nuevo texto. Texto con formato en controles TextView y EditText

Hemos visto cómo crear un objeto Editable y añadir marcas de formato al texto que contiene, pero todo esto no tendría ningún sentido si no pudiéramos visualizarlo. Como ya podéis imaginar, los controles TextView y EditText nos van a permitir hacer esto. Vemos qué ocurre si asignamos al nuestro control EditText el objeto Editable que hemos creado antes:

```
txtTexto.setText(str);
```

Curso Android v2 - 2013 - Todos los Derechos Reservados- Emiliano Gonzalez/Pedro Coronel

Tras ejecutar este código veremos cómo efectivamente en el cuadro de texto aparece el mensaje con el formato esperado:



Ya hemos visto cómo asignar texto con y sin formato a un cuadro de texto, pero ¿qué ocurre a la hora de recuperar texto con formato desde el control?. Ya vimos que la función getText() devuelve un objeto de tipo Editable y que sobre éste podíamos hacer un toString(). Pero con esta solución estamos perdiendo todo el formato del texto, por lo que no podríamos por ejemplo salvarlo a una base de datos. La solución a esto último pasa obviamente por recuperar directamente el objeto Editable y serializarlo de algún modo, mejor aún si es en un formato estándar. Pues bien, en Android este trabajo ya nos viene hecho de fábrica a través de la clase Html [API], que dispone de métodos para convertir cualquier objeto Spanned en su representación HTML equivalente. Veamos cómo. Recuperemos el texto de la ventana anterior y utilicemos el método Html.toHtml(Spannable) para convertirlo a formato HTML:

```
//Obtiene el texto del control con etiquetas de formato HTML

String aux2 = Html.toHtml(txtTexto.getText());
```

Haciendo esto, obtendríamos una cadena de texto como la siguiente, que ya podríamos por ejemplo almacenar en una base de datos o publicar en cualquier web sin perder el formato de texto establecido:

```
<p>Esto es un <b>simulacro</b>.</p>
```

La operación contraria también es posible, es decir, cargar un cuadro de texto de Android (EditText) o una etiqueta (TextView) a partir de un fragmento de texto en formato HTML. Para ello podemos utilizar el método Html.fromHtml(String) de la siguiente forma:

```
//Asigna texto con formato HTML
txtTexto.setText(
Html.fromHtml("<p>Esto es un <b>simulacro</b>.</p>"),
BufferType.SPANNABLE);
```

Desgraciadamente, aunque es de agradecer que este trabajo venga hecho de casa, hay que decir que tan sólo funciona de forma completa con las opciones de formato más básicas, como negritas, cursivas, subrayado o colores de texto, quedando no soportadas otras sorprendentemente básicas como el tamaño del texto, que aunque sí es correctamente¿ traducido por el método toHtml(), es descartado por el método contrario fromHtml(). Sí se

Curso Android v2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

soporta la inclusión de imágenes, aunque esto lo dejamos para más adelante ya que requiere algo más de explicación.

Checkboxes y RadioButtons

Tras hablar de varios de los controles indispensables en cualquier aplicación Android, como son los botones y los cuadros de texto, en este nuevo apartado vamos a ver cómo utilizar otros dos tipos de controles básicos en muchas aplicaciones, los checkboxes y los radio buttons.

Control CheckBox [API]

Un control checkbox se suele utilizar para marcar o desmarcar opciones en una aplicación, y en Android está representado por la clase del mismo nombre, CheckBox. La forma de definirlo en nuestra interfaz y los métodos disponibles para manipularlos desde nuestro código son análogos a los ya comentados para el control ToggleButton. De esta forma, para definir un control de este tipo en nuestro layout podemos utilizar el código siguiente, que define un checkbox con el texto "Márcame":

```
<CheckBox android:id="@+id/ChkMarcame"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Márcame!" />
```

En cuanto a la personalización del control podemos decir que éste extiende [indirectamente] del control TextView, por lo que todas las opciones de formato ya comentadas en capítulos anteriores son válidas también para este control. En el código de la aplicación podremos hacer uso de los métodos isChecked() para conocer el estado del control, y setChecked(estado) para establecer un estado concreto para el control.

```
if (checkBox.isChecked()) {
    checkBox.setChecked(false);
}
```

En cuanto a los posibles eventos que puede lanzar este control, el más interesante es sin duda el que informa de que ha cambiado el estado del control, que recibe el nombre de onCheckedChanged. Para implementar las acciones de este evento podríamos utilizar por tanto la siguiente lógica:

```
final CheckBox cb = (CheckBox)findViewById(R.id.chkMarcame);

cb.setOnCheckedChangeListener(
    new CheckBox.OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView,
                                                    boolean isChecked) {
            if (isChecked) {
```

Curso Android V2 - 2013 - Todos los Derechos Reservados - Emilio J. Gonzalez/Pedro Coronel

```
                cb.setText("Checkbox marcado!");
            }
            else {
                cb.setText("Checkbox desmarcado!");
        }
    }
});
```

Control RadioButton [API]

Al igual que los controles checkbox, un radio button puede estar marcado o desmarcado, pero en este caso suelen utilizarse dentro de un grupo de opciones donde una, y sólo una, de ellas debe estar marcada obligatoriamente, es decir, que si se marca una de ellas se desmarcará automáticamente la que estuviera activa anteriormente. En Android, un grupo de botones RadioButton se define mediante un elemento RadioGroup, que a su vez contendrá todos los elementos RadioButton necesarios. Veamos un ejemplo de cómo definir un grupo de dos controles RadioButton en nuestra interfaz:

```
<RadioGroup android:id="@+id/gruporb"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <RadioButton android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1" />

    <RadioButton android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 2" />

</RadioGroup>
```

Curso Android v4 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez Pedro Coronel

En primer lugar vemos cómo podemos definir el grupo de controles indicando su orientación (vertical u horizontal) al igual que ocurría por ejemplo con un LinearLayout. Tras esto, se añaden todos los objetos RadioButton necesarios indicando su ID mediante la propiedad android:id y su texto mediante android:text. Una vez definida la interfaz podremos manipular el control desde nuestro código java haciendo uso de los diferentes métodos del control RadioGroup, los más importantes: check(id) para marcar una opción determinada mediante su ID, clearCheck() para desmarcar todas las opciones, y getCheckedRadioButtonId() que como su nombre indica devolverá el ID de la opción marcada (o el valor -1 si no hay ninguna marcada). Veamos un ejemplo:

```
final RadioGroup rg = (RadioGroup)findViewById(R.id.gruporb);
rg.clearCheck();
rg.check(R.id.radio1);
int idSeleccionado = rg.getCheckedRadioButtonId();
```

Intents

Un intent sirve para invocar componentes, en android entendemos por componentes las activities, Que son componentes de UI [Interfaz gráfica], services, Código ejecutándose en segundo plano, broadcast receivers, Código que responde a un mensaje de transmisión [Broadcast messages] y proveedores de contenido, código que abstráe los datos.

Introducción a los Intents

Como mecanismo para invocar componentes, los intents son bastante fáciles de comprender. Básicamente nos permiten llamar a aplicaciones externas a la nuestra, lanzar eventos a los que otras aplicaciones puedan responder, lanzar alarmas etc.

Vamos a mostrar un ejemplo, supongamos que tenemos la siguiente activity:

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

El layout R.Layout.MiActivity debe estar declarado y ser un archivo de layout valido. Una vez creado este archivo de layout, es necesario registrarlo en el AndroidManifest, que será algo así:

```
<activity android:name=".MiActivity"
          android:label="Mi Activity">
   <intent -filter>
      <action android:name="nuestra.accion.nombreAccion"/>
      <category android:name="android.intent.category.DEFAULT" />
   </intent>
</activity>
```

Al registrar la activity en el AndroidManifest, registramos también una acción que podremos usar para invocar a dicha actividad. El diseñador de la actividad puede asignar el nombre que crea conveniente a la acción. Ahora que ya está todo listo, podemos lanzar un intent para llamar a esta actividad:

```
public static void invokeMiActivity(Activity activity){
   String actionName= "nuestra.accion.nombreAccion";
   Intent intent = new Intent(actionName);
   activity.startActivity(intent);
}
```

La convención que se usa para nombrar una acción suele ser .intent.action.NOMBRE_ACCION

Curso Android V2 - 2015 Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

Una vez que se invoca a la actividad, ésta tiene la posibilidad de recuperar el intent que la llamó. Y podemos recuperarlo del siguiente modo:

```
//Este código se inserta en el método onCreate() de la actividad.
Intent intent = this.getIntent();
if (intent == null){
   Log.d("Tag", "La actividad no se ha llamado mediante un intent.")
}
```

Intents disponibles en Android

En developer.android.com/guide/appendix/g-app-intents.html se puede encontrar una lista con las aplicaciones disponibles en Android junto con los intents que las invocan. Por ejemplo, para el navegador web, tenemos dos acciones, VIEW y WEB_SEARCH, que abren el navegador en una url específica o realizan una búsqueda.

En el caso del dialer (marcador), tenemos las acciones CALL y DIAL, que vienen dadas por la URI tel:numero_de_teléfono, la diferencia entre estas dos acciones, es que CALL realiza la llamada al número de la URI, y DIAL solo lo marca, pero no realiza la llamada.

Vamos a ver ejemplos de intents que invocan a las aplicaciones mencionadas en la documentación de Android:

```
public static void invokeWebBrowser(Activity activity){
   Intent intent = new Intent(Intent.ACTION_VIEW);
   intent.setData(Uri.parse("http://www.google.com"));
   activity.startActivity(intent);
}


public static void invokeWebSearch(Activity activity){
   Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
   intent.setData(Uri.parse("http://www.google.com"));
   activity.startActivity(intent);
}


public static void dial(Activity activity){
   Intent intent = new Intent(Intent.ACTION_DIAL);
   activity.startActivity(intent);
}


public static void call(Activity activity){
   Intent intent = new Intent(Intent.ACTION_CALL);
   intent.setData(Uri.parse("tel:555-555-555"));
   activity.startActivity(intent);
}

public static void showMapAtLatLong(Activity activity){
   Intent intent = new Intent(Intent.ACTION_VIEW);
   intent.setData(Uri.parse("geo:0,0?z=4&q=restaurantes"));
```

Curso Android V2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez Pedro Coronel

```
    activity.startActivity(intent);
}
```

Composición de los intents

Un intent está formado por una acción, datos (que se representan mediante URIs), datos extra en pares clave/valor y un nombre de clase explícito, llamado nombre del componente.

Es necesario aclarar algo, cuando un intent trae consigo un nombre de componente, se le llama intent explícito. Cuando no lo lleva y depende de la acción y los datos se llama intent implícito.

Intents y Data URIs

Como vimos un poco más arriba los URIs para las acciones ACTION_DIAL y ACTION_CALL tienen la estructura tel:número, y la manera de usar esta URI en el intent para pasarla como dato es la siguiente:

```
intent.setData(Uri.parse("tel:555-555-555"));
```

El nombre de las acciones normalmente suele ser un String o un String constante con el nombre del paquete como prefijo.

La sección de datos de un intent no son datos realmente, se trata de punteros a datos. Está representado por un string que representa una URI. Una Uri de un intent puede contener argumentos, como las urls de las web.

Acciones genéricas

Vamos a volver a ver el código que invoca al navegador web para analizarlo más en profundidad:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
activity.startActivity(intent);
```

En este caso, ACTION_VIEW parece una accíon muy genérica, Android se las ingenia para averiguar a qué actividad llamar en base a esta acción haciendo uso de la composición de la URI. Para ello, mira el esquema que posee la URI, que en este caso es http y pregunta a todas las actividades para saber cual de ellas comprende este esquema. Por lo tanto, la actividad del navegador deberá tener registrada la acción VIEW junto con el esquema de datos de http:

```
<activity ...>
   <intent -filter>
      <action android:name="android.intent.action.VIEW"/>
      <data android:scheme="http" />
      <data android:scheme="https" />
   </intent>
</activity>
```

Curso Android V2 - 2013 - Todos los Derechos Reservados Emiliano Gonzalez/Pedro Coronel

Los intents admiten además de las acciones y datos, un atributo adicional llamado extras. Este tipo de dato viene dado por la forma clave/valor, en la cual el nombre de la clave normalmente suele empezar con el nombre del paquete y el valor puede ser de cualqueira de los tipos fundamentales u objetos arbitrários, siempre que se implemente la interfaz android.os.Parcelable. Esta información extra se representa mediante la clase android.os.Bundle

```
//Obtener un bundle
Bundle b = intent.getExtras();

//Colocar un bundle en un intent
Bundle b2 = new Bundle();

//Rellenar el bundle con datos fundamentales
putExtra(String name, boolean value);
putExtra(String name, int value);
putExtra(String name, double value);
putExtra(String name, String value);

//Otros tipos de datos
putExtra(String name, int[] value);
putExtra(String name, float[] value);
putExtra(String name, Serializable value);
putExtra(String name, Parcelable value);
putExtra(String name, Bundle value);

//Añadir bundles de otros intents
putExtra(String name, Intent otroIntent);

//Añadir el bundle al intent
intent.putExtras(b2)
```

getExtras devuelve el bundle que contenga el intent. Si el intent ya tiene un bundle, putExtras transfiere los pares clave/valor adicionales del bundle nuevo al que ya existía. Si no existe ningun bundle asociado, putExtras creará uno y copiará todos los valores.

La clase intent tiene declarados unas claves extras que acompañan a ciertas acciones, pueden verse en developer.android.com/reference/android/content/Intent.html#EXTRA_ALARM_COUNT. Por ejemplo EXTRA_SUBJECT nos permite almacenar el asunto de un email. El valor de esta clave es android.intent.extra.SUBJECT.

Usar componentes para invocar directamente una activity

Una forma más directa de iniciar una actividad es mediante el su ComponentName, que es una abstracción del nombre del paquete y de la clase. Existen varios métodos para realizar esta acción en la clase Intent:

```
setComponent(ComponentName name);
SeClassName(String packName, String className);
setClassName(Context context, String ClassName);
setClass(Context context, Class classObject);
```

Curso Android 4.2 - 2013 - Todos los Derechos Reservados- Emiliano Gonzalez/Pedro Coronel

Se puede usar el nombre de una clase directamente sin necesidad de construir un ComponentName. Por ejemplo, si tenemos la siguiente actividad:

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

Podemos usar el siguiente código para llamarla:

```
Intent intent = new Intent(activity, MiActivity.class);
activity.start(intent);
```

Así, cualquier intent podrá iniciar la actividad, pero para ello, debemos registrar dicha actividad en el AndroidManifest así:

<activity android:name=".MiActivity" android:label="Mi Activity" />

Sin ningún tipo de intent-filter, ya que estos no son necesarios cuando se invoca a una actividad directamente mediante el nombre de su clase. Recordad que este intent es de tipo explícito.

Las actividades se pueden clasificar en categorías para así poder buscarlas basándonos en el nombre de dicha categoría. Por ejemplo, mientras el sistema se está iniciando, busca en las actividades las que estén bajo la categoría CATEGORY_LAUNCHER.

La convención usada para nombrar a las categorías es (para el caso de CATEGORY_LAUNCHER):

```
android.intent.category.LAUNCHER
```

La forma de declarar las categorías en el AndroidManifest es la siguiente:

```
<activity android:name=".PrincipalActivity"
android:label="@string/app_name">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent>
</activity>
```

Curso Android V2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

Categorias de Activities

Vamos a ver algunas categorías predefinidas, podéis encontrar la lista de todas ellas en developer.android.com/reference/android/content/Intent.html#CATEGORY_ALTERNATIVE:

- CATEGORY_DEFAULT: Si declaramos una actividad bajo la categoría DEFAULT, podrá ser invocada mediante intents implícitos, de lo contrario, habrá que llamarla con intents explícitos.
- CATEGORY_BROWSABLE: Si la actividad es de este tipo, podrá ser invocada con seguridad por el navegador para mostrar los datos referenciados por un link, como una imagen o un email.
- CATEGORY_GADGET: La actividad se puede embeber dentro de otra actividad que pueda albergar gadgets.
- CATEGORY_HOME: Suele existir solo una actividad de este tipo, que es la pantalla principal, esta actividad se muestra al iniciar el teléfono o pulsar el botón home. Si existe más de una se le pregunta al usuario cual elegir.

Cuando usamos un intent para iniciar una actividad podemos especificar qué tipo de actividad queremos especificando la categoría. Otra opción es buscar las actividades que coincidan con una determinada categoría, por ejemplo:

```
Intent i = new Intent(Intent.ACTION_MAIN, null);
i.addCategory(Intent.CATEGORY_LAUNCHER);
PackageManager pm = getPackageManager();
List<resolveinfo> list = pm.queryIntentActivities(i, 0);
</resolveinfo>
```

PackageManager permite encontrar actividades que coincidan con un intent sin llegar a invocarlas. Una vez ejecutado lo de arriba, podemos iterar sobre la lista e invocar a la actividad que coincida con el nombre que deseemos:

```
for(ResolveInfo ri: list){
   Log.d("Info", ri.toString());
   String pkgName = ri.activityInfo.packageName;
   String className = ri.activityInfo.name;

if(className.equals("nombre.paquete.denuestra.actividad.nombreActividad")
){
      Intent i = new Intent();
      i.setClassName(pkgName, className);
      activity.startActivity(i);
   }
}
```

Es posible lanzar una actividad basándonos en el nombre de la categoría:

```
Intent i = new Intent(Intent.ACTION_MAIN, null);
i.addCategory(Intent.CATEGORY_LAUNCHER);
activity.startActivity(i);
```

Como mencioné anteriormente, en el caso de que exista más de una actividad que satisfaga las condiciones que impone el intent, se mostrará un diálogo al usuario para que elija cual lanzar.

Curso Android v2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

Si quisiéramos invocar un intent para volver a la pantalla principal, basta con cambiar la categoría del código de arriba de Intent.CATEGORY_LAUNCHER a Intent.CATEGORY_HOME

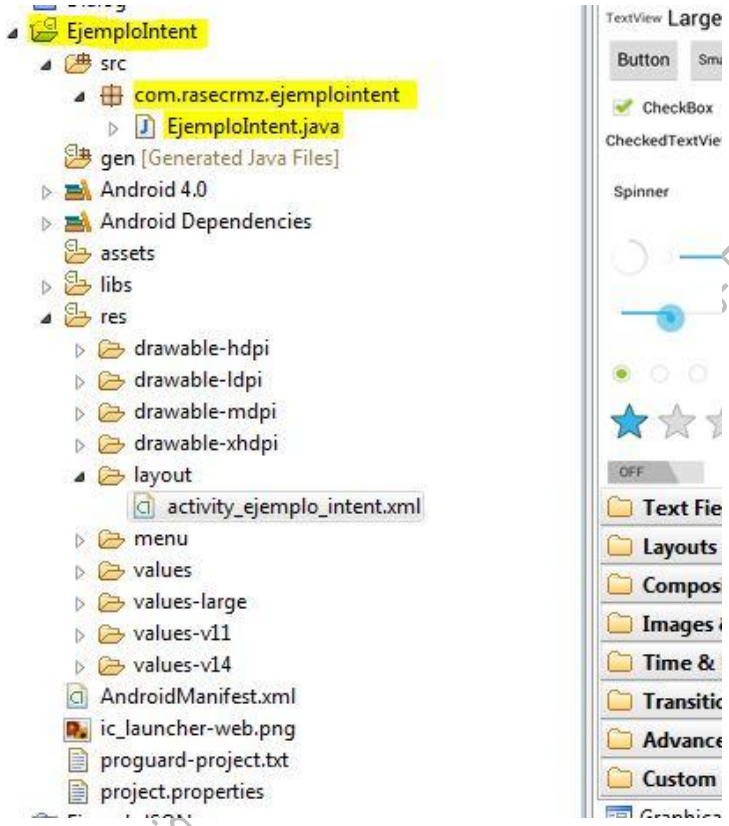
Ejercicio

Crear una aplicación que llame a un nuevo Activity utilizando intents. (Se podría mejorar el ejercicio agregando paso de parámetros entre activities)

Solucion

Paso 1:

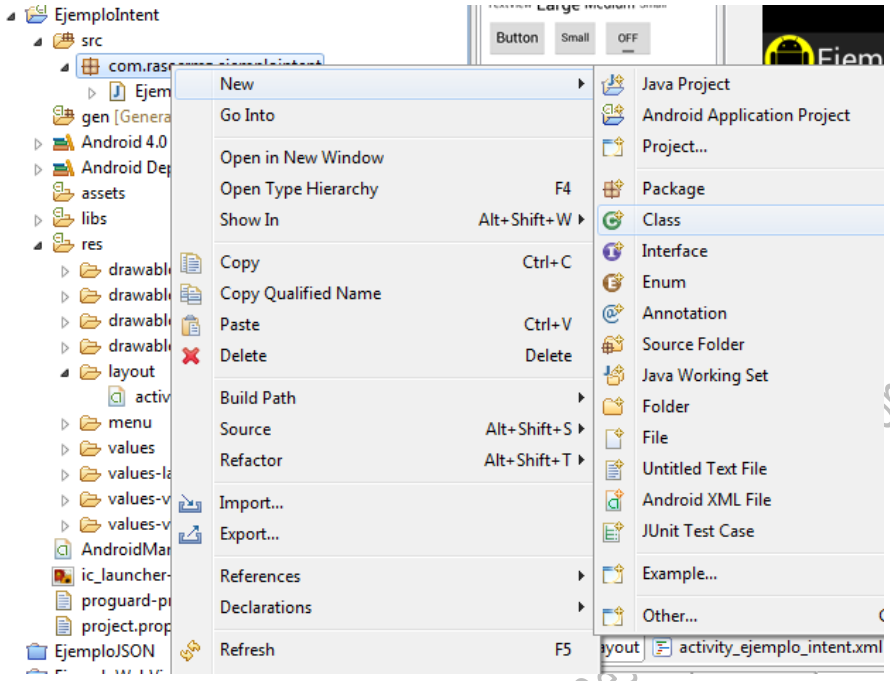
Abrimos Eclipse, y creamos un proyecto nuevo, el cual denominaremos “EjemploIntent”.



Curso Android
Emiliano Gonzalez/Pedro Coronel

Paso 2

Hacer clic derecho en el “package” (dentro de la carpeta src, en “com.rasecrmz.ejemplointent”) y selecciona “Nuevo – Clase”: Nombra la nueva clase “OtraActivity”.



Paso 3:

Abrimos el archivo “AndroidManifest” y lo modificamos para que quede de la siguiente manera:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rasecrmz.ejemplointent"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".EjemploIntent"
            android:label="@string/title_activity_ejemplo_intent" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Curso Android V2 - 2013 - Todos los Derechos Re... onzalez/Pedro Coronel

```
            android:name=".OtraActivity">
            <intent-filter>
                <action android:name="com.rasecrmz.OtraActivity"/>
                <category android:name="android.intent.category.DEFAULT"
/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Paso 4:

Ya que definimos la otra activity en el Manifest, vamos a crear su layout. Has una copia de el archivo activity_ejemplo_intent.xml (o main.xml depende de como lo haya generado eclipse), pega la copia en la misma carpeta layout y ponle nombre “otra_activity_layout.xml”.

Modifica ese archivo para que quede de la siguiente manera:

```
<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="Esta es la Otra Activity"
        tools:context=".EjemploIntent" />

</RelativeLayout>
```

Paso 5:

Modificar el archivo OtraActivity.java de la siguiente manera:

```
package com.rasecrmz.ejemplointent;


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

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

Paso 6:

Curso Android V2 - 2013 - Todos los Derechos Reservados - Emiliano Gonzalez Pablo Coronel

Vamos a agregar un botón a la activity original, para que de esta manera, al dar clic a este boton, se abra la otra activity. Para esto, modificamos el layout de la activity EjemploIntent (en mi caso es activity_ejemplo_intent.xml, comúnmente será 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" >

    <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="Activity Ejemplo"
        tools:context=".EjemploIntent" />
    <Button
         android:id="btn_mostrar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Mostrar la otra activity"
        android:onClick="onClick" />

</RelativeLayout>
```

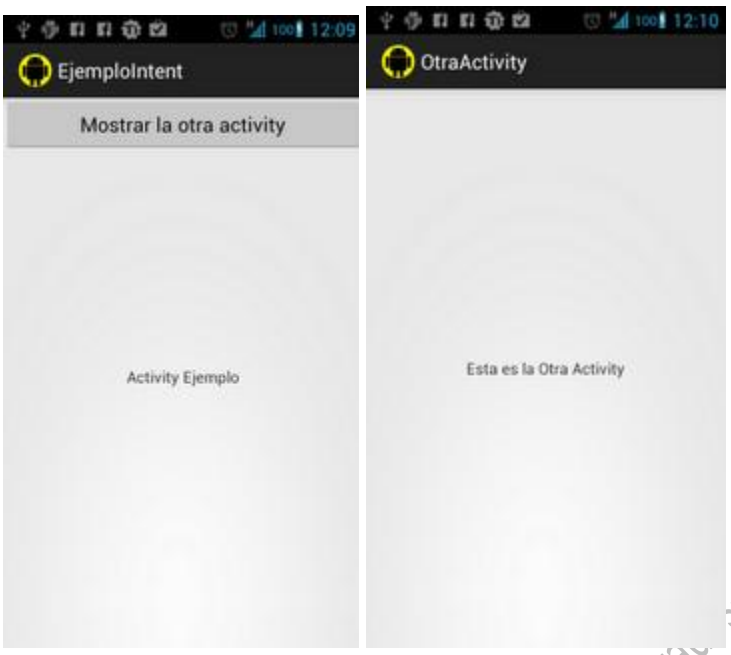
Paso 7:

Agregamos el método onClick a la clase EjemploIntent.java, este método sera invocado cada vez que demos clic en el botón:

```
package com.rasecrmz.ejemplointent;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.support.v4.app.NavUtils;

public class EjemploIntent extends Activity {

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

    public void onClick(View view){
        startActivity(new Intent(this, OtraActivity.class));
    }
}
```

Curso Android v2 - 2015 - Todos los Derechos Reservados - Emiliano Gonzalez/Pedro Coronel

Resultado



Reservados - Emiliano Gonzalez/Pedro Coronel

Comentarios

- Si observan, cuando agregaron la Otra activity en el manifest, estan unos tags denominados intent-filter dentro de la descripción de la activity. El nombre es “com.rasecrmz.OtraActivity”. Otras activities que deseen llamar a esta activity lo hacen utilizando este nombre. La categoria del intent-filter es “android.intent.category.DEFAULT”. Esto es necesario definirlo para que esta activity pueda ser llamada por otras activities usando el metodo startActivity();
- Las activities en android pueden ser invocadas por cualquier aplicacion ejecutándose en el dispositivo. Por ejemplo, puedes crear un nuevo proyecto en Eclipse y desplegar esta “OtraActivity” usando su intent filter.
- Si la activity que estas llamando pertenece al mismo proyecto, puedes llamarla también de esta manera:
- startActivity(new Intent(this, OtraActivity.class));