Interfaz de usuario en Android: Fragments

Fragmentos estáticos:

Los fragments estáticos poseen varias peculiaridades:

- Se declaran en el layout de la actividad mediante la etiqueta fragment.
- No pueden agregarse, remplazarse o eliminarse dinámicamente.

A modo de aplicación de ejemplo para este artículo, nosotros vamos a simular una aplicación de correo adaptándola a tres configuraciones distintas: pantalla "normal" (p.e. un teléfono), pantalla grande horizontal (p.e. tablet) y pantalla grande vertical (p.e. tablet). Para el primer caso colocaremos el listado de correos en una actividad y el detalle en otra, mientras que para el segundo y el tercero ambos elementos estarán en la misma actividad, a derecha/izquierda para el caso horizontal, y arriba/abajo en el caso vertical.

Definiremos por tanto dos fragments: uno para el listado y otro para la vista de detalle. Ambos serán muy sencillos. Al igual que una actividad, cada fragment se compondrá de un fichero de layout XML para la interfaz (colocado en alguna carpeta /res/layout) y una clase java para la lógica asociada.

El primero de los fragment a definir contendrá tan sólo un control `ListView`, para el que definiremos un adaptador personalizado para mostrar dos campos por fila ("De" y "Asunto"). Ya describimos cómo hacer esto en el artículo dedicado al control ListView. El layout XML (lo llamaremos `fragment_listado.xml`) quedaría por tanto de la siguiente forma:

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

```
        android:orientation="vertical" >
        <ListView
            android:id="@+id/LstListado"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >
        </ListView>
    </LinearLayout>
```

Como hemos dicho, todo fragment debe tener asociado, además del layout, su propia clase java, que en este caso debe extender de la clase `Fragment`. Los fragment aparecieron con la versión 3 de Android, por lo que en principio no estarían disponibles para versiones anteriores. Sin embargo, Google pensó en todos y proporcionó esta característica también como parte de la librería de compatibillidad `support-v4`. Si nuestro proyecto incluye la librería de soporte `appcompat-v7` (inlcuida por defecto en los proyectos de Android Studio, ver fichero *build.gradle*) no debemos preocuparnos por más, ya que `support-v4` es una dependencia de ésta y por tanto también estará incluida. En caso de que no utilicemos `appcompat-v7` simplemente tendríamos que añadir la referencia a la librería en la sección de dependencias del fichero *build.gradle* de nuestro módulo principal:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.android.support:support-v4:22.2.1'
}
```

Hecho esto, ya no habría ningún problema para utilizar la clase `Fragment`, y otras que comentaremos más adelante, para utilizar fragmentos compatibles con la mayoría de versiones de Android. Veamos cómo quedaría nuestra clase asociada al fragment de listado.

```
package net.sgoliver.android.fragments;

import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
```

```
10  import android.widget.ListView;
11  import android.widget.TextView;
12
13  public class FragmentListado extends Fragment {
14
15      private Correo[] datos =
16              new Correo[]{
17      new Correo("Persona 1", "Asunto del correo 1", "Texto del correo 1"),
18      new Correo("Persona 2", "Asunto del correo 2", "Texto del correo 2"),
19      new Correo("Persona 3", "Asunto del correo 3", "Texto del correo 3"),
20      new Correo("Persona 4", "Asunto del correo 4", "Texto del correo 4"),
21      new Correo("Persona 5", "Asunto del correo 5", "Texto del correo 5")};
22
23      private ListView lstListado;
24
25      @Override
26      public View onCreateView(LayoutInflater inflater,
27                               ViewGroup container,
28                               Bundle savedInstanceState) {
29
30          return inflater.inflate(R.layout.fragment_listado, container, false);
31      }
32
33      @Override
34      public void onActivityCreated(Bundle state) {
35          super.onActivityCreated(state);
36
37          lstListado = (ListView)getView().findViewById(R.id.LstListado);
38
39          lstListado.setAdapter(new AdaptadorCorreos(this));
40      }
41
42      class AdaptadorCorreos extends ArrayAdapter<Correo> {
43
44          Activity context;
45
46          AdaptadorCorreos(Fragment context) {
47              super(context.getActivity(), R.layout.listitem_correo, datos);
48              this.context = context.getActivity();
49          }
50
51          public View getView(int position, View convertView, ViewGroup parent) {
52              LayoutInflater inflater = context.getLayoutInflater();
53              View item = inflater.inflate(R.layout.listitem_correo, null);
54
55              TextView lblDe = (TextView)item.findViewById(R.id.LblDe);
56              lblDe.setText(datos[position].getDe());
57
58              TextView lblAsunto = (TextView)item.findViewById(R.id.LblAsunto);
59              lblAsunto.setText(datos[position].getAsunto());
60
61              return(item);
62          }
63      }
64  }
```

La clase `Correo` es una clase sencilla, que almacena los campos *De*, *Asunto* y *Texto* de un correo.

```
package net.sgoliver.android.fragments;

public class Correo
{
    private String de;
    private String asunto;
    private String texto;

    public Correo(String de, String asunto, String texto){
        this.de = de;
        this.asunto = asunto;
        this.texto = texto;
    }

    public String getDe(){
        return de;
    }

    public String getAsunto(){
        return asunto;
    }

    public String getTexto(){
        return texto;
    }
}
```

Si observamos con detenimiento las clases anteriores veremos que no existe casi ninguna diferencia con los temas ya comentados en artículos anteriores del curso sobre utilización de controles de tipo lista y adaptadores personalizados. La única diferencia que encontramos aquí respecto a ejemplos anteriores, donde definíamos actividades en vez de fragments, son los métodos que sobrescribimos. En el caso de los fragment son normalmente dos: `onCreateView()` y `onActivityCreated()`.

El primero de ellos, `onCreateView()`, es el "equivalente" al `onCreate()` de las actividades, y dentro de él es donde normalmente asignaremos un layout determinado al fragment. En este caso tendremos que "inflarlo" (convertir el XML en la estructura de objetos java equivalente) mediante el método `inflate()` pasándole como parámetro el ID del layout correspondiente, en nuestro caso `fragment_listado`.

El segundo de los métodos, `onActivityCreated()`, se ejecutará cuando la actividad contenedora del fragment esté completamente creada. En nuestro caso, estamos aprovechando este evento para obtener la referencia al control `ListView` y asociarle su adaptador. Sobre la definición del adaptador personalizado `AdaptadorCorreos` no comentaremos nada porque es idéntico al ya descrito en el artículo sobre listas.

Con esto ya tenemos creado nuestro fragment de listado, por lo que podemos pasar al segundo, que como ya dijimos se encargará de mostrar la vista de detalle. La definición de este fragment será aún más sencilla que la anterior. El layout, que llamaremos `fragment_detalle.xml`, tan sólo se compondrá de un cuadro de texto:

```
1
2       <?xml version="1.0" encoding="utf-8"?>
3       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4           android:layout_width="match_parent"
5           android:layout_height="match_parent"
6           android:orientation="vertical"
7           android:background="#FFBBBBBB" >
8
9           <TextView
10              android:id="@+id/TxtDetalle"
11              android:layout_width="wrap_content"
12              android:layout_height="wrap_content" />
13
14      </LinearLayout>
```

Y por su parte, la clase java asociada, se limitará a inflar el layout de la interfaz. Adicionalmente añadiremos un método público, llamado

`mostrarDetalle()`, que nos ayude posteriormente a asignar el contenido a mostrar en el cuadro de texto.

```
1
2
3       package net.sgoliver.android.fragments;
4
5       import android.os.Bundle;
6       import android.support.v4.app.Fragment;
7       import android.view.LayoutInflater;
8       import android.view.View;
9       import android.view.ViewGroup;
10      import android.widget.TextView;
11
12      public class FragmentDetalle extends Fragment {
13
14          @Override
15          public View onCreateView(LayoutInflater inflater,
16                                   ViewGroup container,
17                                   Bundle savedInstanceState) {
18
19              return inflater.inflate(R.layout.fragment_detalle, container, false);
20          }
21
22          public void mostrarDetalle(String texto) {
23              TextView txtDetalle =
24                  (TextView)getView().findViewById(R.id.TxtDetalle);
25
26              txtDetalle.setText(texto);
          }
      }
```

Una vez definidos los dos fragments, ya tan solo nos queda definir las actividades de nuestra aplicación, con sus respectivos layouts que harán uso de los fragments que acabamos de implementar.

Para la actividad principal definiremos 3 layouts diferentes: el primero de ellos para los casos en los que la aplicación se ejecute en una pantalla normal (por ejemplo un teléfono móvil) y dos para pantallas grandes (por ejemplo una tablet, uno pensado para orientación horizontal y otro para vertical). Todos se llamarán `activity_main.xml`, y lo que marcará la diferencia será la carpeta en la que colocaremos cada uno. Así, el primero de ellos lo colocaremos en la carpeta por defecto `/res/layout`, y los otros dos en las carpetas `/res/layout-large` (pantalla grande) y `/res/latout-`

`large-port` (pantalla grande con orientación vertical) respectivamente. De esta forma, según el tamaño y orientación de la pantalla Android utilizará un layout u otro de forma automática sin que nosotros tengamos que hacer nada.

Para el caso de pantalla normal, la actividad principal mostrará tan sólo el listado de correos, por lo que el layout incluirá tan sólo el fragment `FragmentListado`.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="net.sgoliver.android.fragments.FragmentListado"
    android:id="@+id/FrgListado"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Como podéis ver, para incluir un fragment en un layout utilizaremos una etiqueta`<fragment>` con un atributo `class` que indique la ruta completa de la clase java correspondiente al fragment, en este primer caso "`net.sgoliver.android.fragments.FragmentListado`". Los demás atributos utilizados son los que ya conocemos de `id`, `layout_width` y `layout_height`.

En este caso de pantalla normal, la vista de detalle se mostrará en una segunda actividad, por lo que también tendremos que crear su layout, que llamaremos `activity_detalle.xml`. Veamos rápidamente su implementación:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="net.sgoliver.android.fragments.FragmentDetalle"
    android:id="@+id/FrgDetalle"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Como vemos es análoga a la anterior, con la única diferencia de que añadimos el fragment de detalle en vez de el de listado.

Por su parte, el layout para el caso de pantalla grande horizontal, será de la siguiente forma:

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

    <fragment class="net.sgoliver.android.fragments.FragmentListado"
        android:id="@+id/FrgListado"
        android:layout_weight="30"
        android:layout_width="0px"
        android:layout_height="match_parent" />

    <fragment class="net.sgoliver.android.fragments.FragmentDetalle"
        android:id="@+id/FrgDetalle"
        android:layout_weight="70"
        android:layout_width="0px"
        android:layout_height="match_parent" />

</LinearLayout>
```

Como veis en este caso incluimos los dos fragment en la misma pantalla, ya que tendremos espacio de sobra, ambos dentro de un `LinearLayout` horizontal, asignando al primero de ellos un peso (propiedad `layout_weight`) de 30 y al segundo de 70 para que la columna de listado ocupe un 30% de la pantalla a la izquierda y la de detalle ocupe el resto.

Por último, para el caso de pantalla grande vertical será prácticamente igual, sólo que usaremos un `LinearLayout` vertical.

```
<?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">

    <fragment class="net.sgoliver.android.fragments.FragmentListado"
        android:id="@+id/FrgListado"
        android:layout_weight="40"
        android:layout_width="match_parent"
        android:layout_height="0px" />

    <fragment class="net.sgoliver.android.fragments.FragmentDetalle"
        android:id="@+id/FrgDetalle"
        android:layout_weight="60"
        android:layout_width="match_parent"
        android:layout_height="0px" />
```

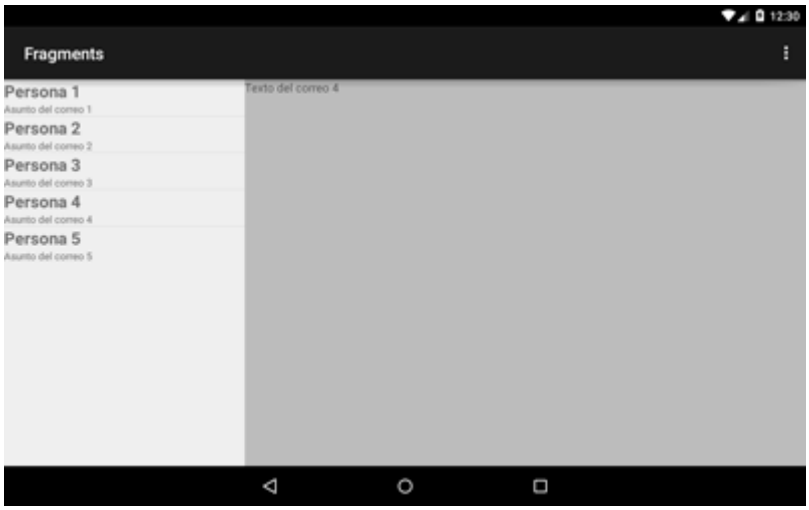
```
17      </LinearLayout>
18
19
20
```

Hecho esto, ya podríamos ejecutar la aplicación en el emulador y comprobar si se selecciona automáticamente el layout correcto dependiendo de las características del AVD que estemos utilizando. En mi caso he definido 2 AVD, uno con pantalla normal y otro grande al que durante las pruebas he modificado su orientación .El resultado fue el siguiente:

Pantalla normal (Nexus 5, de ~5 pulgadas):

Pantalla grande vertical (Nexus 7, de ~7 pulgadas):

Pantalla grande horizontal (Nexus 7, de ~7 pulgadas):



Como vemos en las imágenes anteriores, la interfaz se ha adaptado perfectamente a la pantalla en cada caso, mostrándose uno o ambos fragments, y en caso de mostrarse ambos distribuyéndose horizontal o verticalmente.

Lo que aún no hemos implementado en la lógica de la aplicación es lo que debe ocurrir al pulsarse un elemento de la lista de correos. Para ello, empezaremos asignando el evento `onItemClick()` a la lista dentro del método `onActivityCreated()` de la clase `FragmentListado`. Lo que hagamos al capturar este evento dependerá de si en la pantalla se está viendo el fragment de detalle o no:

1. Si existe el fragment de detalle habría que obtener una referencia a él y llamar a su método `mostrarDetalle()` con el texto del correo seleccionado.
2. En caso contrario, tendríamos que navegar a la actividad secundaria `DetalleActivity` para mostrar el detalle.

Veamos el código.

```
@Override
public void onActivityCreated(Bundle state) {
   super.onActivityCreated(state);

   lstListado = (ListView)getView().findViewById(R.id.LstListado);

   lstListado.setAdapter(new AdaptadorCorreos(this));
```

```
   lstListado.setOnItemClickListener(new OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> list, View view, int pos,
long id) {
         FragmentDetalle fragmento = (FragmentDetalle)
getActivity().getSupportFragmentManager().findFragmentById(R.id.FrgDet
alle);
boolean hayDetalle = (fragmento!=null && fragmento.isInLayout());

 Correo c=(Correo) list.getItemAtPosition(pos);
         if(hayDetalle) {
            fragmento.mostrarDetalle(c.getTexto());
         }
         else {
            Intent i = new Intent(getActivity(),
DetalleActivity.class);
            i.putExtra(DetalleActivity.EXTRA_TEXTO, c.getTexto());
            startActivity(i);
         }
      }
   });
}
```

Un detalle importante a destacar es que la actividad debe heredar de `FragmentActivity` o `AppCompatActivity` (si usamos la librería *appcompat-v7*) para poder hacer uso de fragments.

Un fragment es un objeto independiente de la actividad y puede utilizarse en varias actividades diferentes. Pero un fragment puede comunicarse con la actividad donde esté, accediendo a ella mediante el método **getActivity**().

Por ejemplo:

```
getActivity().findViewById(R.id.action_settings);
```

Una actividad puede comunicarse con un fragment accediendo a el mediante el método **findFragmentById**.

Por ejemplo:

```
getFragmentManager().findFragmentById(R.id.container)
```

En el ejemplo vemos cómo hacemos uso de estos dos métodos, ya que un fragment no tiene relación directa con otros fragments de la misma activity, primero tenemos que recuperar su activity(getActivity()) y desde allí, buscar si existe el otro Fragment (con el SupportFragmentManager).

OnItemClick es el método que se ejecutará cuando el fragment de listado nos avise de que se ha seleccionado un determinado item de la lista. La lógica será la ya mencionada, es decir, si en la pantalla existe el fragment de detalle simplemente lo actualizaremos mediante `mostrarDetalle()` y en caso contrario navegaremos a la actividad `DetalleActivity`. Para este segundo caso, crearemos un nuevo `Intent` con la referencia a dicha clase, y le añadiremos como parámetro extra un campo de texto con el contenido del correo seleccionado. Finalmente llamamos a `startActivity()` para iniciar la nueva actividad.

Y ya sólo nos queda comentar la implementación de esta segunda actividad, `DetalleActivity`. El código será muy sencillo, y se limitará a recuperar el parámetro extra pasado desde la actividad anterior y mostrarlo en el fragment de detalle mediante su método `mostrarDetalle()`, todo ello dentro de su método `onCreate()`.

```
1   package net.sgoliver.android.fragments;
2
3   import android.os.Bundle;
4   import android.support.v7.app.AppCompatActivity;
5
6   public class DetalleActivity extends AppCompatActivity {
7
8       public static final String EXTRA_TEXTO =
9               "net.sgoliver.android.fragments.EXTRA_TEXTO";
10
11      @Override
12      protected void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.activity_detalle);
15
16          FragmentDetalle detalle =
17                  (FragmentDetalle)getSupportFragmentManager()
                        .findFragmentById(R.id.FrgDetalle);

            detalle.mostrarDetalle(getIntent().getStringExtra(EXTRA_TEXTO));
```

```
18          }
19      }
20
21
22
```

Y con esto finalizaríamos nuestro ejemplo de fragments estáticos. Si ahora volvemos a ejecutar la aplicación en el emulador podremos comprobar el funcionamiento de la selección en las distintas configuraciones de pantalla.

Fragmentos Dinámicos

Los fragments dinámicos se agregan dinámicamente a su actividad. Esta operación se realiza sobre los elementos de su vista que están dedicados a acoger el fragment (en general, un FrameLayout). Pueden agregarse, eliminarse o remplazarse durante la ejecución de su actividad.

He aquí una actividad que contiene un FrameLayout en el que se incluye un fragment dinámico.

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

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

    android:id="@+id/container"

    android:layout_width="match_parent"

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

Su fragment es muy sencillo y se corresponde a lo que hemos visto antes:

```
public class PlaceholderFragment extends Fragment {

            public PlaceholderFragment() {

        }




@Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

        View rootView = inflater.inflate(R.layout.fragment_main,

container, false);

            return rootView;

        }

    }
```

A continuación, en el método onCreate de su actividad, debe agregar el fragment a su actividad utilizando el identificador del FrameLayout declarado antes:

```
@Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        getFragmentManager().beginTransaction()

                   .add(R.id.container, new PlaceholderFragment())

                  .commit();

}
```

Cada vez que se agrega, elimina o remplaza un fragment se produce una transacción. Cada transacción contiene las siguientes etapas:

- Recuperación del FragmentManager (getFragmentManager).
- Inicio de la transacción (beginTransaction).
- Procesamiento (en este caso, agregar un fragment): esta operación recibe como parámetro el identificador de la zona donde se agregará el fragment y la instancia del fragment a agregar. También es muy común el uso del método replace (con los mismos parámetros) y remove que recibe el fragment a eliminar.

- Validación de la transacción mediante el método commit. Mientras no se invoque a este método, la transacción no será válida.

Gestión de los fragments

Una transacción realizada sobre un fragment no se agrega a la back stack por defecto, lo que significa que esto no se tendrá en cuenta cuando el usuario haga clic en el botón back.

En ciertos casos, querrá agregar esta transacción a la back stack para que se tenga en cuenta en la navegación del usuario. Par ello, utilice el método **addToBackStack (String name)**, que recibe como parámetro una cadena de caracteres que identifica este elemento de la back stack o bien null en caso contrario.

```
getFragmentManager().beginTransaction()

        .add(R.id.container, new PlaceholderFragment())

        .addToBackStack(null).commit();
```

Puede, también sobrecargar la animación de ejecución de una transacción de fragment mediante los métodos **setTransition** o **setCustomAnimation**.

Por ejemplo:

```
.setCustomAnimations(android.R.anim.fade_in,android.R.anim.fade_out)
```

Gestión de las versones anteriores

Los fragments están disponibles a partir de la versión 3.0 de Android. Para poder gestionar la compatibilidad hacia atrás con las versiones anteriores de Android, utilice la clase Fragment disponible en la biblioteca v4.

Debe operar de modo que su actividad herede de la clase **FragmentActivity** y utilizar el método **getSupportFragmentManager** en lugar de la clase **getFragmentManager**.