CONCEPTO DE FRAGMENT Y SU UTILIZACIÓN EN ANDROID

Ángel Rodríguez Duport

Programación Multimedia y de Dispositivos Móviles

10/02/2014

Índice de contenidos

CONCEPTO DE FRAGMENT Y SU UTILIZACIÓN EN ANDROID

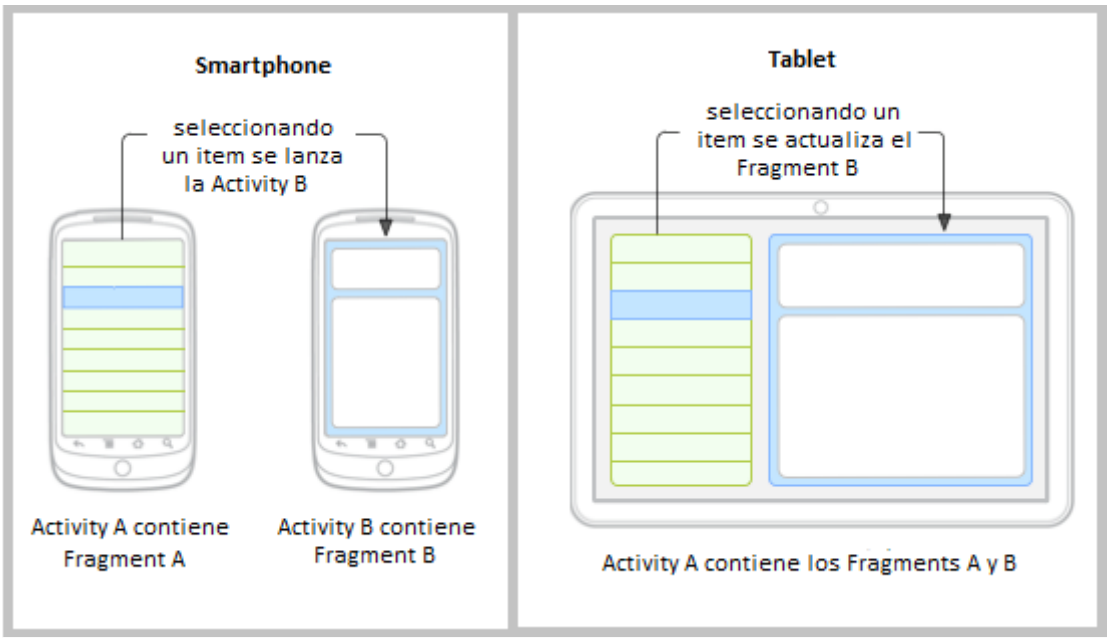
1.Introducción

En Android, una Activity normalmente ocupa el tamaño de una pantalla de Smartphone, mostrando los layouts que generan la interfaz gráfica de usuario en una aplicación. Es decir, la Activity es un contenedor para las vistas.

Pero nos encontramos con un problema estético cuando una Activity se muestra en un dispositivo con pantalla más grande, como una tablet. No porque no pueda mostrarse, sino que como la pantalla es más grande, los layouts intentarán hacer uso del espacio sobrante y esto genera cambios en la jerarquía de los layouts.

Para solucionar esto se hace uso de "mini activities", las cuales contienen su propio set de layouts. Una vez ejecutada, una Activity puede tener una o mas "mini activities" en base a la orientación de la pantalla del dispositivo en uso. Desde Android 3.0 estas "mini activities" son denominadas fragments.

A groso modo un fragment funciona como una Activity, ya que los fragments contienen layouts al igual que las anteriores, pero la diferencia es que los fragments siempre están dentro de una Activity.



2.Concepto de Fragment

Vamos a definir fragment como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma totalmente independiente del resto de elementos de la Activity, y que como es obvio puede reutilizarse en otras activities. Esto nos permite poder dividir la interfaz en varias porciones, de tal modo que podamos diseñar distintas configuraciones de pantalla en base a su orientación y tamaño.

Un fragment es una "pieza" de software en la que definimos un determinado diseño visual y un comportamiento asociado a la misma, y que además, tiene la ventaja de que puede trabajar con otros fragments en una misma Activity.

3.Características

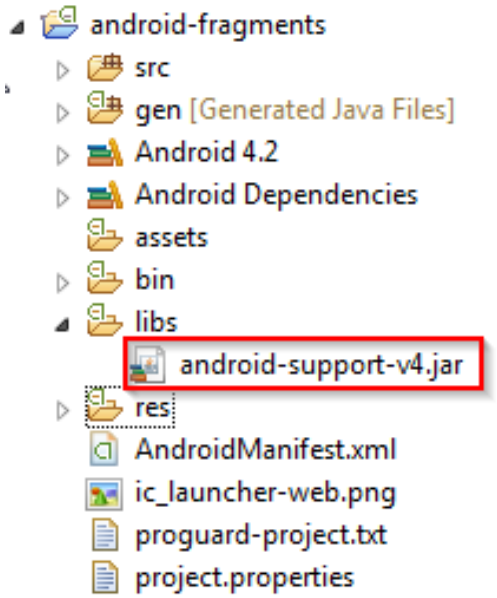
Ya hemos mencionado el concepto de fragment y algunas de sus características, estas son las más importantes:

- Cada fragment tiene su propia interfaz individual, es decir, lleva asociado su propio layout.
- Las clases java asociadas a los fragment heredan o tienen como superclase a Fragment (extends Fragment).
- Cada uno de los fragment que se declaren constará de su propio código y actuará de forma independiente al resto.
    - Un fragment tiene un ciclo de vida propio.
    - Un fragment no conoce la existencia de otros fragment.
- Un fragment necesita de una Activity para que esta actúe de contenedor y para intercambiar datos con otros fragment. Esta Activity tiene como superclase a FragmentActivity (extends Fragment Activity).

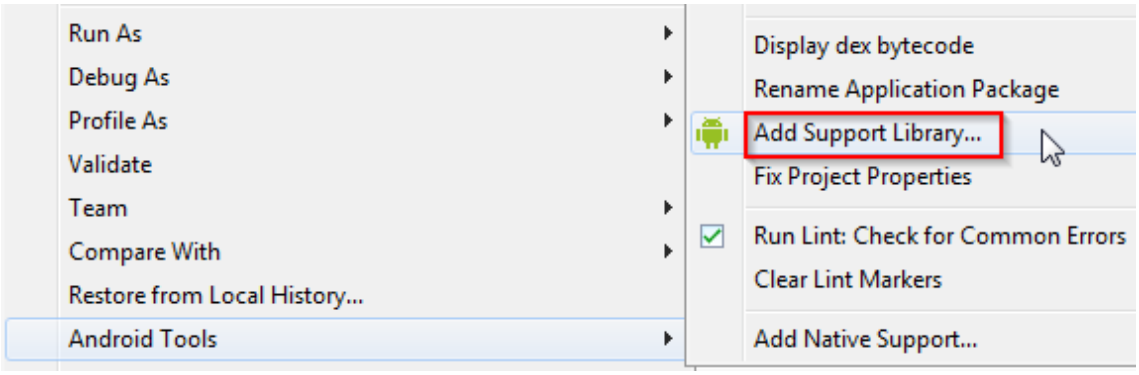
4.Librerías necesarias

Dicho queda que todo fragment debe tener asociada, además del layout, su propia clase java, y que esta clase debe heredar de Fragment. Es aquí donde puede surgir algún problema, ya que los fragment aparecieron con la versión 3 de

Android y eso da lugar a que en principio no estarían disponibles para versiones anteriores. Dicho esto, sabemos que Google proporciona está característica como parte de la librería de compatibilidad Android-Support. No nos vamos a encontrar con este problema en las versiones más recientes del plugin de Android para Eclipse ya que en este entorno se añade por defecto a todos los proyectos creados como se puede ver en la siguiente imagen.



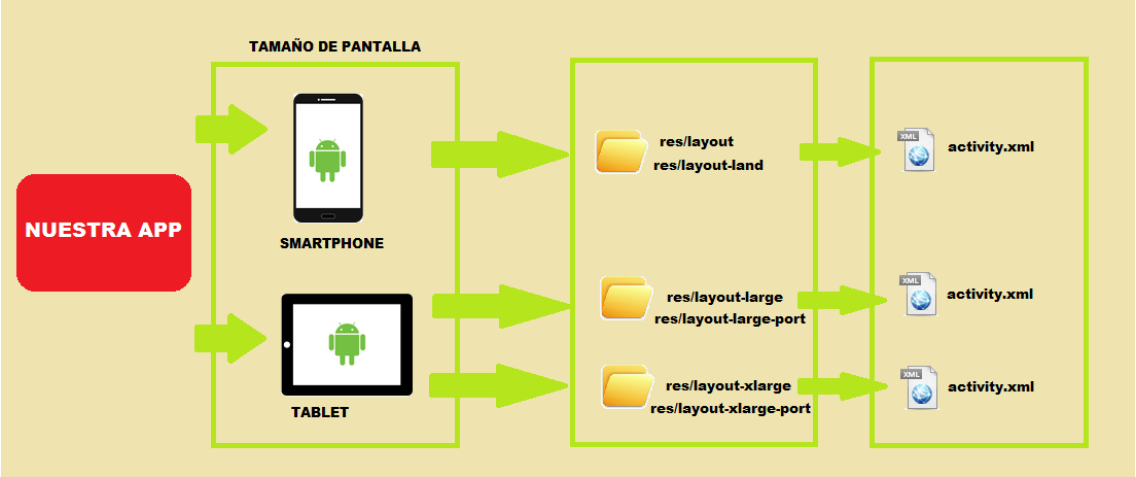
Si por alguna circunstancia esto no ocurriese, el .jar podría importarse de forma manual en el proyecto que queramos mediante la opción “ADD SUPPORT LIBRARY” que se encuentra en el menú contextual del proyecto como podemos apreciar en la siguiente imagen.



Dicho esto no tendríamos ningún problema para empezar a programar nuestros fragments ya que tenemos a nuestra disposición la clase Fragment.

5. Layouts y configuración de pantalla

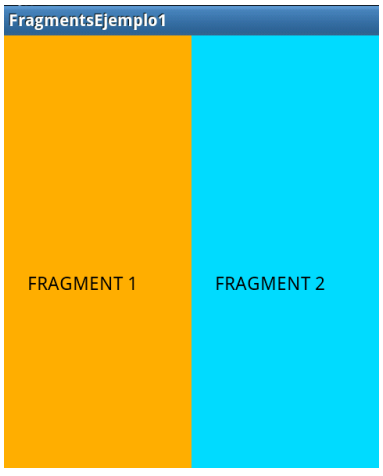
Hemos dicho que los fragments surgieron a partir del aumento del tamaño de la pantalla de los dispositivos móviles para optimizar el espacio y mejorar la estética. Pues bien, Android nos ofrece la posibilidad de mostrar una misma aplicación en distintos dispositivos sin tener que cambiar código, simplemente añadiendo carpetas con ficheros .xml que el dispositivo reconocerá y ejecutará según su tamaño. Dejo una imagen explicativa de donde hay que generar estas carpetas.

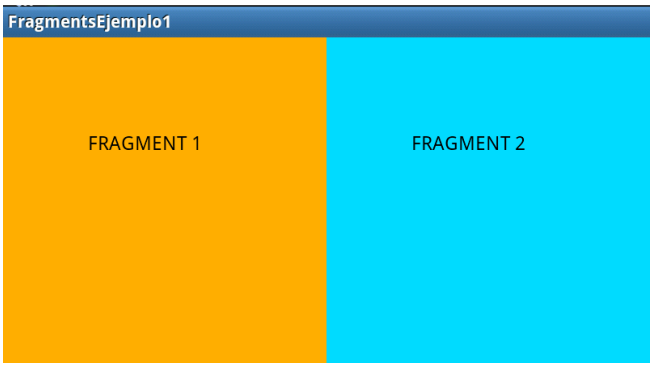


En los Smartphones colocaremos nuestro layout en la carpeta **res/layout**, y si queremos definir la configuración de la orientación horizontal crearemos la carpeta res**/layout-land.** Para dispositivos de mayor tamaño como las tablets tenemos **res/layout-large** (horizontal) **y res/layout-large-port** (vertical) para tablets de mediano tamaño, y para tablets más grandes **res/layout-xlarge** (horizontal) **y res/layout-xlarge-port** (vertical)

6. Un ejemplo sencillo para entender el funcionamiento

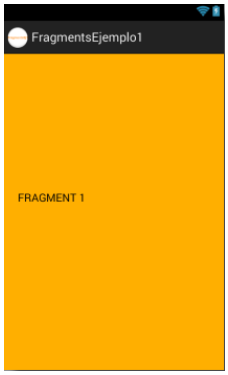
Vamos a ver un ejemplo muy sencillo de cómo crear una Activity cuyo layout asociado contenga dos fragments con sus respectivas clases asociadas. La salida final sería algo así:





1. El primer paso es como no puede ser de otra forma crear un proyecto nuevo de una aplicación Android. Como considero que todo el mundo llegado a este punto sabe hacerlo lo voy a pasar por alto.

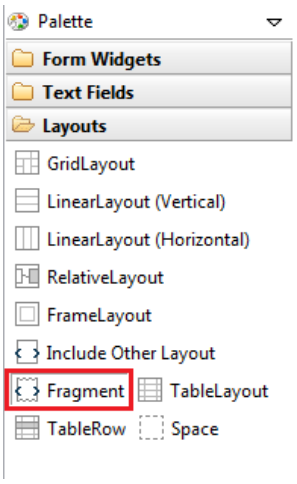
2. Una vez creado nuestro proyecto y definidas nuestra MainActivity(extends FragmentActivity) y su layout correspondiente pasamos a crear dos layouts que corresponderán a la vista de cada uno de los fragments. Para ello nos desplazamos a la carpeta **res/layout** y con el botón derecho **New… /Android XML File**. Crearemos dos archivos XML y los llamaremos fragment 1.xml y fragment2.xml. Para el ejemplo simplemente le he agregado un TextView y un texto que indique a que fragment pertenece. También le he dado un color para distinguir bien entre uno y otro.

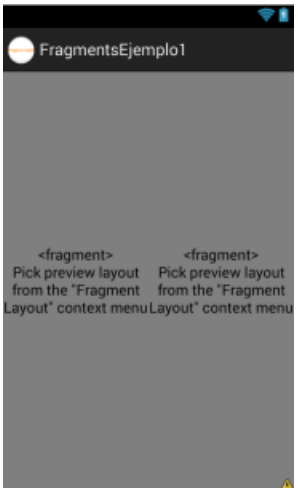


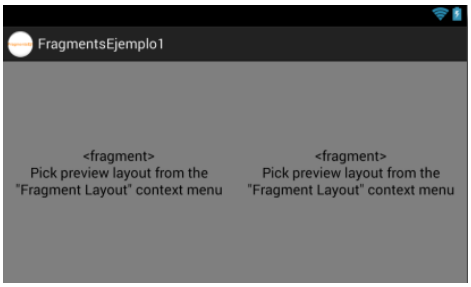
3. Cuando hayamos creado los layouts nos toca crear dos clases que extiendan de Fragment y asociarles estos dos layouts.
   Sobre el Package botón derecho del ratón, New… Class, y creamos dos clases Fragment1 y Fragment2. Una vez hecho esto definimos la superclase (extends Fragment) en las dos clases creadas y por último les asociamos un layout a cada una de ellas… En la siguiente captura se ve la clase Fragment1, la otra sería igual solo que cambiando el layout.

```
1  package com.angelrodriguezduport.fragmentsejemplo1;
2
3  import android.os.Bundle;
8
9  public class Fragment1 extends Fragment {
10
11     @Override
12     public View onCreateView(LayoutInflater inflater, ViewGroup container,
13             Bundle savedInstanceState) {
14         return inflater.inflate(R.layout.fragment1, container, false);
15     }
16
17 }
```

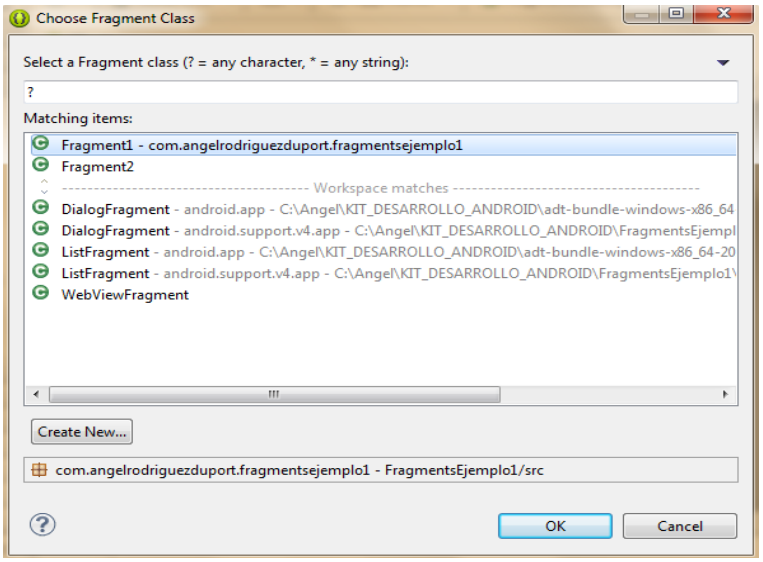
4. Cuando hayamos creado las dos clases y hayamos asociado los layouts a estas nos desplazamos a el layout.xml de la MainActivity y agregamos dos componentes fragment que se encuentran en **Palette/Layouts/Fragment**.







Cuando introducimos un componente fragment debemos seleccionar la clase a la que va asociado. Nos saldrá una pantalla como la siguiente en la que seleccionaremos la clase deseada:

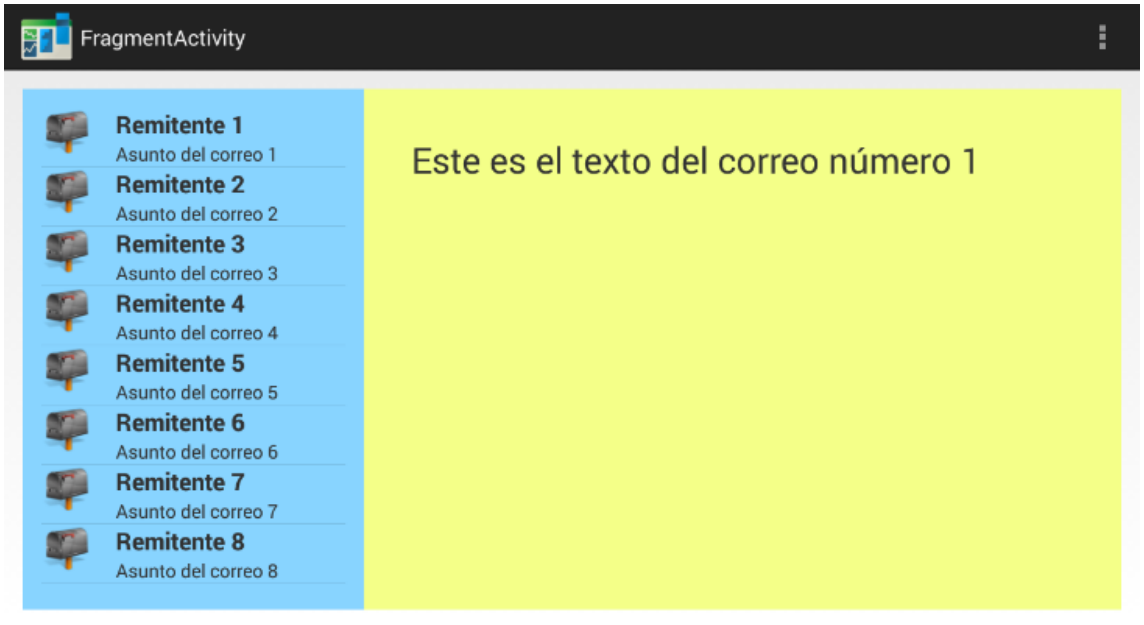


Nada más, ya podemos probar la aplicación. Realmente no hace nada, ya que no estamos comunicanco un fragment con otro ni con ninguna Activity, simplemente refleja de forma básica el funcionamiento de los Fragment.

En el próximo ejercicio podemos ver de forma más detallada el uso de los fragment y la comunicación con otros fragments y Activities.

7. Aplicación desarrollada: Uso de fragments en una app de gestión de correo electrónico

El objetivo es el desarrollo de una App que a través de dos fragment nos permita la visualización de correos electrónicos. Para ello mostraremos en un fragment una lista con los correos electrónicos y en otro fragment el detalle o contenido de este. La vista final de la App sería así:



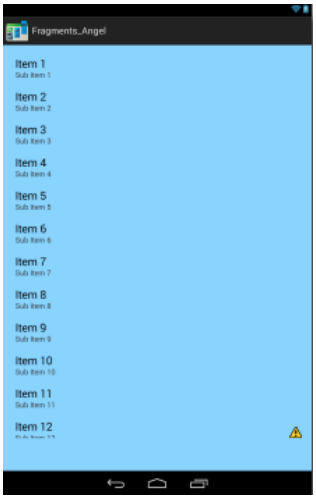
Definiremos por tanto dos fragments: uno para el listado y otro para la vista de detalles. Ambos serán muy sencillos. Al igual que una actividad, cada fragment se compondrá de un fichero de layout XML para la interfaz (colocado en alguna carpeta `/res/layout`) y una clase java para la lógica asociada.

También definiremos una clase Activity(extends FragmentActivity) que contenga los fragments y otras clases que nos harán falta y que iremos creando a medida que las vayamos necesitando.

Empezamos creando la clase CORREO.java:

```
public class Correo implements Serializable{

    private String remitente;
    private String asunto;
    private String contenido;

    public Correo(String remitente, String asunto, String contenido) {
        this.remitente = remitente;
        this.asunto = asunto;
        this.contenido = contenido;
    }

    public String getRemitente() {
        return remitente;
    }

    public String getAsunto() {
        return asunto;
    }

    public String getContenido() {
        return contenido;
    }
}
```

Luego el primero de los layouts(`fragment_lista.xml`) que irá asociado al fragment que contendrá el listado de correos llevará tan sólo un ListView, para el que definiremos un adaptador personalizado(`listitem_correo.xml`) para mostrar dos campos por fila (“Remitente” y “Asunto”).



`fragment_lista.xml`

Ahora creamos nuestra clase asociada al fragment de la lista de correos: FRAGMENTLISTA.java. Esta clase heredará de Fragment y se encargará de crear objetos Correo:

```
package com.angelrodriguezduport.fragments_angel;

import android.app.Activity;

public class FragmentLista extends Fragment {

    private Correo[] correos = new Correo[] {
            new Correo("Remitente 1", "Asunto del correo 1",
                    "Este es el texto del correo número 1"),
            new Correo("Remitente 2", "Asunto del correo 2",
                    "Este es el texto del correo número 2"),
            new Correo("Remitente 3", "Asunto del correo 3",
                    "Este es el texto del correo número 3"),
            new Correo("Remitente 4", "Asunto del correo 4",
                    "Este es el texto del correo número 4"),
            new Correo("Remitente 5", "Asunto del correo 5",
                    "Este es el texto del correo número 5"),
            new Correo("Remitente 6", "Asunto del correo 6",
                    "Este es el texto del correo número 6"),
            new Correo("Remitente 7", "Asunto del correo 7",
                    "Este es el texto del correo número 7"),
            new Correo("Remitente 8", "Asunto del correo 8",
                    "Este es el texto del correo número 8") };;

    private ListView lVListadoCorreos;
    private CorreosListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_lista, container, false);
    }
```

El método `onCreateView()`, es el equivalente al `onCreate()` de las actividades, es decir, que dentro de él es donde normalmente asignaremos un layout determinado al fragment. En este caso tendremos que "inflarlo" mediante el método `inflate()` pasándole como parámetro el ID del layout correspondiente, en nuestro caso `fragment_lista.xml`. Seguimos codificando la clase con otro método propio de los Fragment.

```
    public void onActivityCreated(Bundle state) {
        super.onActivityCreated(state);
        lVListadoCorreos = (ListView) getView().findViewById(R.id.listView1);
        lVListadoCorreos.setAdapter(new MyAdapterCorreo(this,correos));

        lVListadoCorreos.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> list, View view, int pos,
                    long id) {
                if (listener != null) {
                    listener.onCorreoSeleccionado((Correo) lVListadoCorreos
                            .getAdapter().getItem(pos));
                }
            }
        });
    }

    public interface CorreosListener {
        void onCorreoSeleccionado(Correo c);
    }

    public void setCorreosListener(CorreosListener listener) {
        this.listener = listener;
    }
}
```

El método `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 (terminada la explicación de esta clase veremos su código).

Se puede observar que hay definida una interfaz `CorreosListener` y un método `setCorreosListener()`. Esto lo vamos a usar para comunicar el fragment con la Activity Principal. Veremos un poco más adelante la utilidad de esta interfaz ya que en este momento desconocemos el elemento externo que va a recurrir a esta interfaz.

Pasamos a definir el código del adaptador del ListView, para ello creamos la clase MYADAPTERCORREO.java. No voy a comentar nada al respecto del código que contiene el adaptador, porque simplemente enlaza el ListView con el layout definido anteriormente llamado `listitem_correo.xml` y asignarle el array de objetos Correo creado en la clase `FragmentLista.java`:

```java
package com.angelrodriguezduport.fragments_angel;

import android.app.Activity;

public class MyAdapterCorreo extends ArrayAdapter<Correo> {

    Activity context;
    Correo[] correos;

    MyAdapterCorreo(Fragment context, Correo[] correos) {
        super(context.getActivity(), R.layout.listitem_correo, correos);
        this.context = context.getActivity();
        this.correos = correos;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = context.getLayoutInflater();
        View item = inflater.inflate(R.layout.listitem_correo, null);

        TextView remitente = (TextView) item
                .findViewById(R.id.textViewRemitente);
        remitente.setText(correos[position].getRemitente());

        TextView lblAsunto = (TextView) item
                .findViewById(R.id.textViewAsunto);
        lblAsunto.setText(correos[position].getAsunto());

        return (item);
    }

}
```

Ya tenemos la parte del listado, ahora vamos a por la parte del detalle o la descripción de los correos. Para ello empezamos creando otro layout, que llamaremos `fragment_detalle.xml`, tan sólo se compondrá de un cuadro de texto (no cuelgo captura de pantalla porque el TextView lo he puesto sin texto y la imagen simplemente sería un layout con fondo amarillo).

Ahora como hicimos para el fragment del listado debemos crear una clase Fragment que se le asocie a este último layout. Esta clase la llamaremos FRAGMENTDETALLE.java y contendrá los métodos propios de un fragment como ya vimos anteriormente. Inflaremos `fragment_detalle.xml` y adicionalmente añadiremos un método público, llamado mostrarDetalle(), que nos ayude posteriormente a asignar el contenido a mostrar en el cuadro de texto. El código que contendrá esta clase es el que se muestra a continuación:

```java
package com.angelrodriguezduport.fragments_angel;

import android.os.Bundle;

public class FragmentDetalle extends Fragment {
    private TextView tVDetalle;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_detalle, container, false);
    }

    public void mostrarDetalle(String texto) {
        tVDetalle = (TextView) getView().findViewById(
                R.id.textViewFragmentDetalle);

        tVDetalle.setText(texto);
    }

}
```

Cuando hayamos terminado de definir los dos fragments, tan solo nos hace falta definir las actividades de las que hará uso nuestra aplicación, con sus respectivos layouts que utilizarán los fragments que acabamos de codificar.

Vamos a pasar a codificar el layout de la Activity que contendrá los layout. En este caso el layout asociado se llama `activity_fragment.xml`. Vimos en el punto 6 de este documento como se agregaban dos fragment a un layout, utilizaremos los pasos indicados en este punto y enlazaremos cuando nos lo pida cada fragment con su clase Fragment correspondiente (`FragmentListado.java` y `FragmentDetalle.java`).

Una vez que este generado el layout vamos a codificar la clase `ActivityFragments`. Pero antes viene la explicación de la lógica de implementar una interfaz en la clase `FragmentListado`, ya que ahora se puede entender.

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.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.

En caso contrario, tendríamos que navegar a la actividad secundaria `ActivityDetalle` para mostrar el detalle.

Sin embargo existe un problema, un fragment no tiene por qué conocer la existencia de ningún otro, es más, deberían diseñarse de tal forma que fueran lo más independientes posible, de forma que puedan reutilizarse en distintas situaciones sin problemas de dependencias con otros elementos de la interfaz.

Por este motivo, el patrón utilizado normalmente en estas circunstancias no será tratar el evento en el propio fragment, sino definir y lanzar un evento personalizado al pulsarse el item de la lista y delegar a la actividad contenedora la lógica del evento, ya que ella sí debe conocer qué fragments componen su interfaz. Esto lo hacemos de forma análoga a cuando definimos eventos personalizados para un control. Definimos una interfaz con el método asociado al evento, en este caso llamada `CorreosListener` con un único método llamado `onCorreoSeleccionado()`, declaramos un atributo de la clase con esta interfaz y definimos un método set...() para poder asignar el evento desde fuera de la clase, todo esto lo codificamos anteriormente en la clase `FragmentListado`.

Volvemos a la clase `FragmentListado y vemos que` lo único que deberemos hacer en el evento `onItemClick()` de la lista será lanzar nuestro evento personalizado `onCorreoSeleccionado()` pasándole como parámetro el contenido del correo, que en este caso obtendremos accediendo al adaptador con `getAdapter()` y recuperando el elemento con `getItem()`.

Hecho esto, el siguiente paso será tratar este evento en la clase java de nuestra actividad principal. Para ello, en el `onCreate()` de nuestra actividad, obtendremos una referencia al fragment mediante el método `getFragmentById()` del fragment manager (componente encargado de gestionar los fragments) y asignaremos el evento mediante su método `setCorreosListener()` que acabamos de definir.

Vamos a hacer que nuestra actividad herede de nuestra interfaz `CorreosListener`, por lo que nos basta pasar `this` al método `setCorreosListener()`. Adicionalmente, un detalle importante a descatar es que la actividad no hereda de la clase `Activity` como de costumbre, sino de `FragmentActivity`.

Código de ACTIVITYFRAGMENTS.java:

```
package com.angelrodriguezduport.fragments_angel;
import com.angelrodriguezduport.fragments_angel.FragmentLista.CorreosListener;

public class ActivityFragments extends FragmentActivity implements CorreosListener {

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

        FragmentLista frgLista = (FragmentLista) getSupportFragmentManager()
                .findFragmentById(R.id.fragment1);
        frgLista.setCorreosListener(this);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    @Override
    public void onCorreoSeleccionado(Correo correo) {
        boolean siExisteElFragmentDetalle = (getSupportFragmentManager().findFragmentById(
                R.id.fragment2) != null);

        if (siExisteElFragmentDetalle) {
            ((FragmentDetalle) getSupportFragmentManager().findFragmentById(
                    R.id.fragment2)).mostrarDetalle(correo.getContenido());
        } else {
            Intent i = new Intent(this, ActivityDetalle.class);
            i.putExtra("contenido", correo.getContenido());
            startActivity(i);
        }
    }
```

La mayor parte del interés de la clase anterior está en el método `onCorreoSeleccionado()`. Éste es el método que se ejecutará cuando el fragment de lista nos avise de que se ha seleccionado un determinado item de la lista. Esta vez sí, 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 `ActivityDetalle`. 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 tan sólo nos queda comentar la implementación de esta segunda actividad, `ActivityDetalle`. 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()`.

```java
package com.angelrodriguezduport.fragments_angel;


import android.os.Bundle;

public class ActivityDetalle extends FragmentActivity {


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

        FragmentDetalle detalle =
                (FragmentDetalle)getSupportFragmentManager()
                    .findFragmentById(R.id.fragment2);

        detalle.mostrarDetalle(getIntent().getStringExtra("correo"));
    }
}
```

Y ya está acabado, ya podemos probar la aplicación después de tanta explicación de código. Cabe destacar que la mayor dificultad está en entender el funcionamiento de la interface, ya que el resto es ampliación de lo explicado en el punto 6 y no tiene gran dificultad de implantación si se sigue paso a paso.

8. Problemas encontrados

No he encontrado muchas dificultades ni en la asimilación del concepto de fragment ni en la implementación del código. Los únicos problemas que he encontrado han venido a la hora de codificar la aplicación del punto 7, a la hora de establecer la comunicación entre los fragments. La solución ha sido acudir a http://www.sgoliver.net/ y entender a base de darle muchas vueltas la solución que se plantea en este sitio web a través de una interface. Para explicarlo correctamente tengo que decir que parte de la explicación del tema 7 ha sido tomada de aquí.

9. Bibliografía

Concepto:

- http://fsvelectronicainformatica.blogspot.com.es/2013/01/que-son-los-fragments-fragmentos-en.html
- http://jarroba.com/fragments-fragmentos-en-android/

Ejemplos:

- http://www.limecreativelabs.com/fragments-en-android/
- http://sekthdroid.wordpress.com/2013/02/01/introduccion-fragments-en-android/
- http://elbauldeandroid.blogspot.com.es/2013/12/fragments.html

Ayuda en la implementación de App Gestión correos(punto 7):

- http://www.sgoliver.net/blog/?p=3496