25.- Fragments

Cuando empezaron a aparecer **dispositivos de gran tamaño tipo tablet**, el equipo de Android tuvo que solucionar el problema de la adaptación de la interfaz gráfica de las aplicaciones a ese nuevo tipo de pantallas. Una interfaz de usuario diseñada para un teléfono móvil no se adaptaba fácilmente a una pantalla varias pulgadas mayor. La solución a esto vino en forma de un nuevo tipo de componente llamado **Fragment**.

Un *fragment* no puede considerarse ni un *control* ni un *contenedor*, aunque se parecería más a lo segundo. Un ***fragment*** podría definirse como **una porción de la interfaz de usuario que puede añadirse o eliminarse de la interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades**. Esto, aunque en principio puede parecer algo trivial, nos va a permitir poder dividir nuestra interfaz en varias porciones de forma que podamos diseñar diversas configuraciones de pantalla, **dependiendo de su tamaño y orientación**, sin tener que duplicar código en ningún momento, sino tan sólo utilizando o no los distintos fragmentos para cada una de las posibles configuraciones. Intentemos aclarar esto un poco con un ejemplo.

Supongamos una aplicación de correo electrónico, en la que por un lado debemos mostrar la lista de correos disponibles, con sus campos clásicos *De* y *Asunto*, y por otro lado debemos mostrar el contenido completo del correo seleccionado. En un teléfono móvil lo habitual será tener una primera actividad que muestre el listado de correos, y cuando el usuario seleccione uno de ellos se navegue a una nueva actividad que muestre el contenido de dicho correo. Sin embargo, en una tablet puede existir espacio suficiente para tener ambas partes de la interfaz en la misma pantalla, por ejemplo en una tablet en posición horizontal podríamos tener una columna a la izquierda con el listado de correos y dedicar la zona derecha a mostrar el detalle del correo seleccionado, todo ello sin tener que cambiar de actividad.

Antes de existir los fragments podríamos haber hecho esto implementando diferentes actividades con diferentes layouts para cada configuración de pantalla, pero esto nos habría obligado a duplicar gran parte del código en cada actividad. Tras la aparición de los fragments, colocaríamos el **listado de correos en un fragment y la vista de detalle en otro**, cada uno de ellos acompañado de su lógica de negocio asociada, y tan sólo nos quedaría definir varios layouts para cada configuración de pantalla donde se incluyeran [o no] cada uno de estos fragments.

A modo de aplicación de ejemplo para este apartado/punto, nosotros vamos a simular la aplicación de correo que hemos comentado antes, 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 se vió cómo hacer esto en el apartado/punto dedicado al control **ListView**. El layout XML (lo llamaremos **fragment_listado.xml**) quedaría por tanto de la siguiente forma:

```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">

    <ListView
        android:id="@+id/lstListado"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
 </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 compatibilidad `support-v4`.** Si nuestro proyecto incluye la librería de soporte `appcompat` (incluida 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` simplemente tendríamos que añadir la referencia a la librería en la sección de dependencias del fichero ***build.gradle*** de nuestro módulo principal:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

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 com.example.ejfragments;

import android.app.Activity;
import android.os.Bundle;
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;
import android.widget.ListView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
```

```
public class FragmentListado extends Fragment {

    private Correo[] datos = new Correo [] {
       new Correo ("Persona 1", "Asunto del correo 1", "Texto del Correo 1"),
       new Correo ("Persona 2", "Asunto del correo 2", "Texto del Correo 2"),
       new Correo ("Persona 3", "Asunto del correo 3", "Texto del Correo 3"),
       new Correo ("Persona 4", "Asunto del correo 4", "Texto del Correo 4"),
       new Correo ("Persona 5", "Asunto del correo 5", "Texto del Correo 5"),
       new Correo ("Persona 6", "Asunto del correo 6", "Texto del Correo 6"),
       new Correo ("Persona 7", "Asunto del correo 7", "Texto del Correo 7")
    };

    private ListView lstListado;
    private CorreoListener listener;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_listado, container, false);
    }

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        lstListado = (ListView)getView().findViewById(R.id.lstListado);
        lstListado.setAdapter(new AdaptadorCorreos(this));

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

    class AdaptadorCorreos extends ArrayAdapter<Correo> {
        Activity context;

        AdaptadorCorreos(Fragment context) {
            super(context.getActivity(), R.layout.listitem_correo, datos);
            this.context = context.getActivity();
        }

        @NonNull
        @Override
        public View getView(int position,
                            @Nullable View convertView,
                            @NonNull ViewGroup parent) {
```

```
                LayoutInflater inflater = context.getLayoutInflater();
                View item = inflater.inflate(R.layout.listitem_correo, null);

                TextView lblDe = (TextView) item.findViewById(R.id.lblDe);
                lblDe.setText(datos[position].getDe());

                TextView lblAsunto = (TextView)item.findViewById(R.id.lblAsunto);
                lblAsunto.setText(datos[position].getAsunto());
                return (item);
            }
        }

    }
```

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

```
package com.example.ejfragments;

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 texto(){
        return texto;
    }
}
```

listitem_correo.xml

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

```
    <TextView android:id="@+id/lblDe"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp" />

    <TextView android:id="@+id/lblAsunto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="normal"
        android:textSize="12sp" />
</LinearLayout>
```

Si observamos con detenimiento las clases anteriores veremos que no existe casi ninguna diferencia con los temas ya comentados en apartados/puntos 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:

```
<?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"
    android:background="#FFBBBBBB">

    <TextView
        android:id="@+id/txtDetalle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="25dp"/>
        android:textStyle="bold" />

</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.

```
package com.example.ejfragments;

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

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;

public class FragmentDetalle extends Fragment {
    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_detalle, container,false);
    }

    public void mostrarDetalle (String texto) {
        TextView txtDetalle =
                (TextView)getView().findViewById(R.id.txtDetalle);
        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="com.example.ejfragments.FragmentListado"
    android:id="@+id/frgListado"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Como se puede comprobar, 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 "`com.example.ejfragments.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 su implementación:

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

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

Añadimos un nuevo recurso `activity_main.xml` en una nueva carpeta `layout-large`

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="com.example.ejfragments.FragmentListado"
        android:id="@+id/frgListado"
        android:layout_weight="30"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

    <fragment
        class="com.example.ejfragments.FragmentDetalle"
        android:id="@+id/frgDetalle"
        android:layout_weight="70"
        android:layout_width="0dp"
        android:layout_height="match_parent" />

</LinearLayout>
```

Como se puede comprobar 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.

Añadimos un nuevo recurso `activity_main.xml` en una nueva carpeta `layout-large-port`

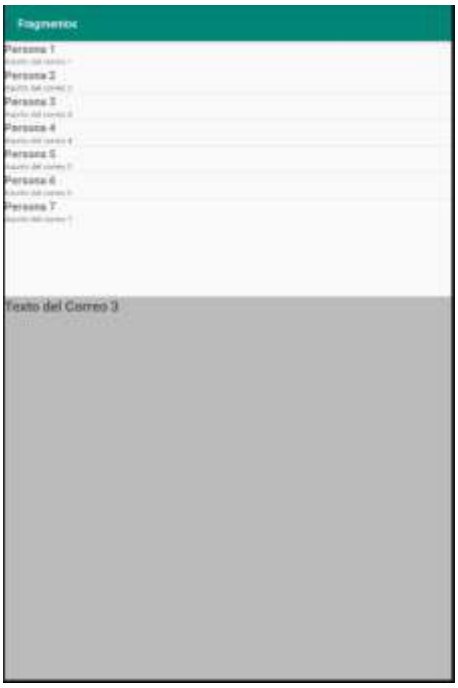
Por último, para el caso de pantalla grande vertical será prácticamente igual, sólo que usaremos un `LinearLayout` vertical.

```
<?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="com.example.ejfragments.FragmentListado"
        android:id="@+id/frgListado"
        android:layout_weight="40"
        android:layout_width="match_parent"
        android:layout_height="0px"/>

    <fragment
        class="com.example.ejfragments.FragmentDetalle"
        android:id="@+id/frgDetalle"
        android:layout_weight="60"
        android:layout_width="match_parent"
        android:layout_height="0px"/>

</LinearLayout>
```

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. Para probarlo, definimos 2 AVD, uno con pantalla normal (El que veníamos utilizando hasta ahora) y otro grande al que durante las pruebas modificaremos su orientación. El resultado es el siguiente:

Pantalla normal (Nexus 5X, de ~5,2” pulgadas):

Pantalla grande vertical (tablet Nexus 10 de 10,05” pulgadas):



Pantalla grande horizontal (tablet Nexus 10 de 10,05” 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.

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. ¿Cómo hacemos esto? Pues 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 clase con esta interfaz y definimos un método `setXXXListener()` (`setCorreoListener()`) para poder asignar el evento desde fuera de la clase. Veamos cómo quedaría:

```
public class FragmentListado extends Fragment {

    . . .
    private CorreosListener listener;

    . . .

@Override
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    lstListado = (ListView)getView().findViewById(R.id.lstListado);
    lstListado.setAdapter(new AdaptadorCorreos(this));

    //Asignamos el evento onItemClick() a la lista de los correos
    lstListado.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent,
                                View view, int position, long id) {
            if (listener != null)
                listener.onCorreoSeleccionado(
                        (Correo)lstListado.getAdapter().getItem(position));
        }
    });
}
```

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

    public void setCorreoListener (CorreoListener listener){
        this.listener =listener;
    }
}
```

Como vemos, una vez definida toda esta parafernalia, 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 `setCorreoListener()` que acabamos de definir.

```
package com.example.ejfragments;

import androidx.appcompat.app.AppCompatActivity;

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

public class MainActivity extends AppCompatActivity  implements CorreoListe-
ner {

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

        FragmentListado fragmentListado = (FragmentListado)
               getSupportFragmentManager().findFragmentById(R.id.frgListado);
        fragmentListado.setCorreoListener(this);
    }


   @Override
   public void onCorreoSeleccionado(Correo c) {
        boolean hayDetalle = (getSupportFragmentManager().
                             findFragmentById(R.id.frgDetalle)!= null);

        if (hayDetalle) {
            ((FragmentDetalle)getSupportFragmentManager().
                         findFragmentById(R.id.frgDetalle)).
                                    mostrarDetalle(c.getTexto());
        }
        else {
            Intent i = new Intent(this, DetalleActivity.class);
```

```
                i.putExtra(DetalleActivity.EXTRA_TEXTO, c.getTexto());
                startActivity(i);
            }
        }
    }
```

Como vemos en el código anterior, en este caso hemos hecho 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 debe heredar de `FragmentActivity` o `AppCompatActivity` (si usamos la librería *appcompat-v7*) para poder hacer uso de fragments.

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 listado 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 `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()`.

```
package com.example.ejfragments;

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

import androidx.appcompat.app.AppCompatActivity;

public class DetalleActivity extends AppCompatActivity {

   public static final String EXTRA_TEXTO =
                        "com.example.ejfragments.EXTRA_TEXTO";

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

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

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

}
```

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

Actividad propuesta: Fragments

Realizar una aplicación con Fragmentos con libro, peliculas, discos, ... tu eliges. De tal forma que tengamos un listado y al seleccionar uno, aparezca la información detallada del elemento seleccionado, si es un libro información detallada de un libro, si es una pelicula de la pelicula, ...