Capítulo 9
Fragments

Práctica 9.1: Dos actividades: detalles de los libros

En esta práctica vamos a añadir una segunda actividad a nuestra aplicación de la lista de libros, de manera que si el usuario pulsa sobre un libro se muestre, en una segunda actividad, su resumen. Seguiremos sin preocuparnos de la fuente de esos datos, y cablearemos los resúmenes también en el código fuente.

1. Haz una copia de la práctica 8.14.
2. Renombra el paquete principal a DM2E.mislibros.
3. Cambia el título de la aplicación en el fichero de cadenas por Mis libros.
4. Elimina el evento que asociamos para borrar los libros.
5. Modifica la clase Libro para que guarde una tercera cadena con el supuesto resumen del libro.

```
class Libro {
        public Libro(String t, String a, String r) {
                titulo = t;
                autor = a;
                resumen = r;
        }
        public String titulo;
        public String autor;
        public String resumen;
} // class Libro
```

6. No es necesario modificar el patrón ViewHolder, dado que el resumen no lo vamos a mostrar en la lista.
7. Modifica la inicialización del ArrayList para añadir un “resumen” (valdrá añadir cualquier cosa que te parezca).

```
libros.add(new Libro("Don Quijote de la Mancha",
"Miguel de Cervantes", "En un lugar de la mancha, de cuyo nombre no quiero acordarme..."));
```

8. Crea una actividad nueva (usando la plantilla “Blank activity”). Llámala ResumenLibroActivity, asociada al layout activity_resumen_libro.
1. Como título pon “Resumen del libro”. No la marques como actividad para el lanzador.
9. Comprueba que en el manifiesto se ha incluido la declaración de la nueva actividad.
10. Elimina la parte de configuración del menú, borrando los métodos, el XML del menú y las constantes de tipo cadena. Elimina también la cadena de hello_world que no usaremos.
11. Modifica el layout para que incluya tres etiquetas, una para el título del libro, otra para el autor, y una tercera para el resumen. Pon

identificadores en las tres para poder acceder a ellas posteriormente. Por comodidad, utiliza un LinearLayout en vertical en lugar de un RelativeLayout. Haz que las tres etiquetas ocupen todo el espacio a lo ancho, y que la del resumen ocupe todo el espacio a lo alto. Pon, como ayuda, un texto “place holder” en cada etiqueta para ver un ejemplo de texto y que sea más fácil intuir el resultado.

```
<LinearLayout
...
android:orientation="vertical">
<TextView android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/tituloLibro"
style="@style/Base.TextAppearance.AppCompat.Title"
android:text="TituloPlaceHolder"/>
<TextView android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/autorLibro"
android:text="AutorPlaceHolder"
style="@style/Base.TextAppearance.AppCompat.Subhead"/>
<TextView android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/resumenLibro"
android:text="Resumen" />
</LinearLayout>
```

12. Añade dos métodos (protegidos) ‘setLibro()’ y ‘setLabel()’ en la actividad ResumenLibroActivity, que reciba las tres cadenas, y configure las tres etiquetas.

```
protected void setLibro(String titulo, String autor,
String resumen) {
setLabel(R.id.tituloLibro, titulo);
setLabel(R.id.autorLibro, autor);
setLabel(R.id.resumenLibro, resumen);
} // setLibro

protected void setLabel(int id, String str) {
TextView tv;
tv = (TextView) findViewById(id);
if (str != null)
tv.setText(str);
else
tv.setText("");
} // setLabel
```

13. En la clase MainActivity, modifica el evento asociado a la pulsación de un elemento de la lista. Hasta ahora mostrábamos un aviso para indicar que se podía quedar pulsando para borrarlo. Ahora lo que queremos es lanzar la segunda actividad para ver el resumen del libro. Necesitamos un Intent que haga una invocación explícita a la carga de la segunda actividad, indicando directamente su clase:

```
lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View view,
```

```
int position, long id) {
Intent intent = new Intent(getApplicationContext(),
ResumenLibroActivity.class);
startActivity(intent);
}
});
```

14. Desde aquí querríamos enviar los datos del libro seleccionado a la segunda actividad. Por desgracia no somos nosotros quienes construimos el objeto de la clase, por lo que no podemos llamar al método setLibro() que hicimos antes. Hay que utilizar el bundle del intent. Añade, antes de la llamada a startActivity(intent):

```
Libro l = aa.getItem(position);
intent.putExtra("titulo", l.titulo);
intent.putExtra("autor", l.autor);
intent.putExtra("resumen", l.resumen);
```

15. Lo que hemos hecho es meter en “la tabla hash” de comunicación con la segunda actividad el título, el autor y el resumen utilizando como índice las cadenas título, autor y resumen. En la clase de la actividad secundaria, modifica el método onCreate(...) y añade después del setContentView(...):

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
setLibro(extras.getString("titulo"),
extras.getString("autor"),
extras.getString("resumen"));
}
```

16. Prueba la aplicación. Comprueba que si pulsas sobre un libro, se abre la segunda ventana con los detalles del libro.
17. Haber utilizado directamente cadenas como claves en el bundle no es muy limpio. Declara en ResumenLibroActivity tres constantes (public static final) de tipo String, y utilizalas tanto en los putExtra() como en el getString().

```
public static final String TITULO = "titulo";
public static final String AUTOR = "autor";
public static final String RESUMEN = "resumen";

//Dentro de onCreate
        …
setLibro(extras.getString(TITULO),
extras.getString(AUTOR),
extras.getString(RESUMEN));
```

Práctica 9.2: Repasando el ciclo de vida

Antes de seguir, vamos a repasar el ciclo de vida de las actividades. Vamos a crear una nueva actividad manualmente (sin layout) en la que sobrescribiremos todos los métodos del ciclo de vida para mostrar un mensaje de log.

Luego haremos que las dos actividades que hicimos en la práctica anterior hereden de ella, de modo que nos informen en cada evolución de su estado. Probaremos luego la aplicación para observar cuando suceden.

Las figuras muestran el ciclo de vida de las actividades tal y como se describe en la documentación de Android.

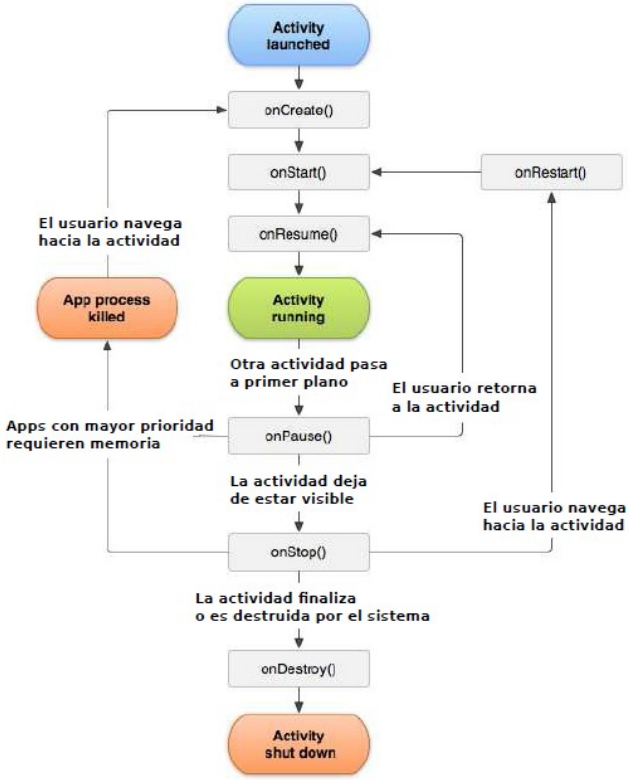

Figura 2.1: Bucles del ciclo de vida de una actividad

1. Haz una copia del proyecto de la práctica anterior.
2. Haz una nueva clase Java y llámala SpyActivity. No utilices el asistente de creación de actividades: no queremos que el IDE nos cree automáticamente el XML del layout, ni añada cadenas, o modifique el fichero de manifiesto.

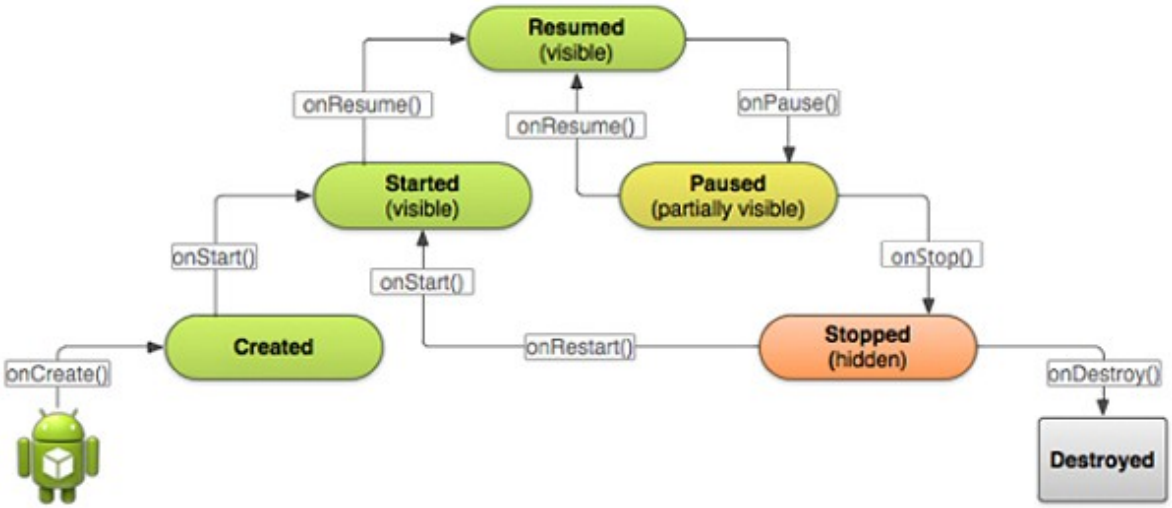

Figura 2.2: Diagrama de estados de una actividad

3. Haz que la nueva clase herede de AppCompatActivity, la superclase de nuestras actividades.
4. Captura todos los eventos del ciclo de vida, llama al método de la superclase y escribe en el log una nota de la invocación.

```
public class SpyActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
if (savedInstanceState == null)
android.util.Log.i(TAG, "onCreate()");
else
android.util.Log.i(TAG, "onCreate(" +
savedInstanceState + ")");
} // onCreate
@Override
protected void onStart() {
android.util.Log.i(TAG, "onStart()");
super.onStart();
}
@Override
protected void onRestart() {
android.util.Log.i(TAG, "onRestart()");
super.onRestart();
}
@Override
protected void onResume() {
android.util.Log.i(TAG, "onResume()");
super.onResume();
}
@Override
protected void onPause() {
android.util.Log.i(TAG, "onPause()");
super.onPause();
}
@Override
protected void onStop() {
android.util.Log.i(TAG, "onStop()");
super.onStop();
}
@Override
protected void onDestroy() {
android.util.Log.i(TAG, "onDestroy()");
super.onDestroy();
}
//-----------------------------------------------
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
android.util.Log.i(TAG, "onSaveInstanceState(" +
savedInstanceState + ")");
super.onSaveInstanceState(savedInstanceState);
}
@Override
protected void onRestoreInstanceState
```

```
(Bundle savedInstanceState) {
android.util.Log.i(TAG, "onRestoreInstanceState(" +
savedInstanceState + ")");
super.onRestoreInstanceState(savedInstanceState);
}
//-----------------------------------------
private static final String TAG = "MainActivity";
} // MainActivity
```

5. La Clase Principal MainActivity hereda de AppCompatActivity cambia y que herede de SpyActivity

Lee cada una de las siguientes pruebas y trata de predecir qué mensajes se van a generar. Luego comprueba si has acertado y trata de explicar las discrepancias.

6. Lanza la aplicación.
7. Pulsa el botón de volver (escape en el teclado).
8. Busca la aplicación instalada y vuelve a lanzarla.
9. Pulsa sobre el botón para ir al “escritorio” (no el botón de volver).
10. Busca la aplicación instalada y vuelve a “lanzarla”.
11. Gira el dispositivo. En un AVD se consigue, por ejemplo, con el número 7 del teclado numérico si no está activado, o con la combinación Ctrl + F11 (o Ctrl + F12).
12. Pulsa sobre cualquier libro.
13. Pulsa la tecla “Volver”.
14. Vuelve a pulsar sobre cualquier libro.
15. Gira de nuevo el dispositivo.
16. Pulsa “Volver”.
17. Pulsa de nuevo sobre cualquier libro.
18. Gira el dispositivo dos veces.
19. Pulsa “Volver”.
20. Cierra la aplicación pulsando una última vez “Volver”.

Práctica 9.3: Fragmentos: aprovechando la pantalla de las tablets

El patrón de actividad con una lista y actividad con los detalles de un elemento es muy habitual en Android. Pero no queda bien en pantallas grandes.

1. Crea un nuevo AVD de tipo tablet. Por eficiencia, procura evitar resoluciones excesivamente grandes como mucha densidad de pantalla. Por ejemplo, crea una tablet antigua, de 7"WSVGA (600x1024, mdpi).
2. Lanza el AVD, y ponlo en orientación apaisado.
3. Lanza la aplicación de la práctica anterior sobre el nuevo AVD, y observa el aparente desperdicio de espacio. En el lado derecho podríamos perfectamente ver el resumen del libro seleccionado.

Para resolver este problema aparecieron los fragmentos, que no es más que una porción de un interfaz de usuario. La ventaja de los fragments es la

versatilidad: podemos crear un fragmento y decidir en ejecución dónde incrustarlo, y en qué circunstancias.

En esta práctica vamos a hacer uso de fragments para tener un interfaz modulable. De momento asumiremos que nuestra aplicación se ejecutará siempre un una tablet apaisada, de modo que no nos preocuparemos de conseguir que funcione bien en móviles.

Lo primero que vamos a hacer es convertir a la actividad secundaria en un fragmento, y la incorporaremos en la otra, temporalmente, para ver el resultado.

1. Haz una copia del proyecto de la práctica anterior.
2. Renombra la clase ResumenLibroActivity por ResumenLibroFragment.
3. Renombra el layout del fichero activity_resumen_libro.xml y llámalo fragment_resumen_libro.xml. Observa cómo se actualiza automáticamente el uso del recurso R en el onCreate().
4. Modifica la superclase (ResumenLibroFragment) para que en lugar de heredar de SpyActivity herede de Fragment. Dado que esta clase apareció en Android 3.0 por primera vez, si queremos que nuestra aplicación funcione en versiones anteriores tendremos que heredar de la reimplementación en la librería de compatibilidad, en el paquete android.support.v4.app..
5. Aunque no es importante, crea un constructor por defecto (sin parámetros) y sin código. Siempre debe haber un constructor por defecto (al que llamará Android por nosotros).
6. Elimina del fichero de manifiesto (AndroidManifest.xml) la entrada de la actividad.

Los fragmentos tienen su propio ciclo de vida, integrado en el de la actividad en el que se conectan. La vista que contienen deben devolverla como resultado de la ejecución de su método onCreateView(...), en lugar de establecerla en el onCreate() como tenemos ahora. El método onCreateView() tiene varios parámetros:

- LayoutInflater: un objeto capaz de construir una vista a partir de un recurso de tipo layout. Nos lo pasan como parámetro ya configurado para que no tengamos que conseguirlo nosotros. Los fragmentos no son contextos (como las actividades).
- ViewGroup: contenedor donde nos incluirán. Lo usaremos al llamar al método inflate para que pueda obtener información sobre el espacio disponible.
- Bundle: información “persistente” para reconstruir el contenido.

Fíjate que el último parámetro no es el que utilizábamos en la actividad con getIntent().getExtras(), dado que ese era el que se había establecido al hacer la invocación explícita. Ahora no van a poder lanzarnos así (ya no somos una actividad), por lo que de momento nos olvidamos de la inicialización del contenido.

1. Elimina el onCreate() y escribe el nuevo método, onCreateView(). Es importante que no incluyas la vista que construyas en el padre, porque lo hará Android por nosotros.

```
public View onCreateView(LayoutInflater inflater,
        ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_resumen_libro,
        container, false);
}
```

2. Los fragmentos no tienen el método findViewById(), por lo que el método setLabel() tendrá un error. El TextView tenemos que buscarlo en nuestra vista. Modifica la linea del error:

```
tv = (TextView) getView().findViewById(id);
```

3. Ejecuta la aplicación. Pulsa sobre cualquier elemento de la lista y comprueba que la aplicación falla. ¿Por qué? ¿Por qué no lo hizo antes?

Lo que queremos ahora es incluir el fragmento que acabamos de hacer dentro de la ventana de la lista. Este paso lo desharemos más adelante, pero nos sirve para comprender y probar la filosofía de los fragmentos. Actualmente el layout de la actividad contiene un RelativeLayout en la raíz con la lista y los dos botones. Lo que queremos es que ese RelativeLayout sea un elemento más, que tenga a su derecha al fragmento. Vamos a hacer que la lista utilice un tercio de la pantalla, y el fragmento del resumen utilice los otros dos tercios.

1. Abre el fichero activity_main, utilizando la vista del XML.
2. Encierra el contenido que aparece dentro de un LinearLayout que será el nuevo nodo raíz. Mueve las definiciones de los namespaces que tiene el RelativeLayout (atributos xmlns:android y xmlns:tools al nuevo nodo raíz.
3. Configura el LinearLayout para que ocupe todo el espacio disponible y tenga una disposición (android:orientation) horizontal.
4. Modifica la configuración del RelativeLayout para poner como ancho 0dp y como layout_weight un 1. Para que el peso se aplique el ancho debe ser 0. El peso se sumará con el peso del resto de nodos hermanos, y se repartirá en función de su tamaño relativo.
5. Añade un nuevo nodo, hermano del RelativeLayout original que sea el fragment. Pon también como ancho 0dp, y como layout_weight un 2 (el doble que para la lista). Establece la clase que contiene el fragmento.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal">
<RelativeLayout
android:layout_weight="1"
android:layout_width="0dp"
....>
[ ... CONTENIDO ORIGINAL ... ]
</RelativeLayout>
```

```
<fragment
android:id="@+id/resumenFragment"
android:layout_width="0dp" android:layout_weight="2"
android:layout_height="match_parent"
android:name="libro.azul.mislibros.ResumenLibroFragment"/>
</LinearLayout>
```

6. Lanza la aplicación en la tablet. Observa el resultado.

Si ejecutas la aplicación en un móvil, verás que la disposición se mantiene, pero al haber mucho menos hueco el resultado es agobiante. Si pones el móvil en apaisado, el aspecto es algo más razonable.

Si queremos que reaccione ante la pulsación de un libro, ya no podemos usar un intent, porque no tenemos una actvidad. Además, tampoco podemos utilizar findViewById() porque eso nos daría una vista, y no queremos tener que lidiar nosotros con ella. Lo que queremos es llamar al método setLibro() del objeto de la clase ResumenLibroFragment que se ha creado para nuestro fragmento.

Para eso, cada actividad tiene un gestor de fragmentos que almacena los fragmentos que se están mostrando. Si estás utilizando una versión posterior al API level 10, la actividad tendrá el método getFragmentManager() que nos lo devuelve.

Sin embargo, aquí estamos utilizando todo el tiempo la librería de compatibilidad, y las clases "nativas"no son intercambiables con ellas. Nuestro fragmento no es un android.app.Fragment, sino un objeto de la clase android.support.v4.app.Fragment, y por tanto no es gestionado por el gestor devuelto por getFragmentManager(), sino por el devuelto por getSupportFragmentManager().

En el código de la pulsación de un elemento de la lista pon el código siguiente:

```
lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
                Libro l = aa.getItem(position);
                ResumenLibroFragment rlf;
                rlf = (ResumenLibroFragment) getSupportFragmentManager().
                findFragmentById(R.id.resumenFragment);
                rlf.setLibro(l.titulo, l.autor, l.resumen);
        }
});
```

Ahora toca convertir en un fragmento también la lista, y luego tendremos una actividad con un layout nuevo que tendrá dos los fragmentos.

1. Deshaz los cambios en el layout de la actividad principal, para que no tenga el fragmento del resumen del libro, ni el LinearLayout. Quita también el layout_weight del RelativeLayout, y vuelve a poner en match_parent el ancho.

2. Dado que ahora el layout de la actividad ya no tiene el fragmento, el código en el evento de la pulsación de un item ha dejado de compilar. Quita el código.
3. Renombra el fichero del layout 'activity_main' a fragment_lista_libros.xml. Observa cómo el IDE nos ha cambiado automáticamente la referencia a él en el onCreate() de la, de momento, actividad.
4. Renombra la clase MainActivity por ListaLibrosFragment. Observa cómo cambia, entre otras muchas cosas, el fichero de manifiesto.
5. Haz que ListaLibrosFragment herede de Fragment. De nuevo, utilizaremos android.support.v4.app.Fragment de la librería de compatibilidad.
6. En el fichero de manifiesto, elimina la referencia a la "actividad", que ya no es tal. Habremos dejado a la aplicación sin ninguna actividad.
7. Como antes, ahora tenemos que construir la vista sobreescribiendo el método onCreateView(), no en el onCreate(). Además, los fragmentos no son contextos, por lo que no podemos usarlo para construir el adaptador:

```
@Override
public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState) {
        // Creamos la lista de libros.
        // [ ... código original de inicialización de datos ... ]
        View result = inflater.inflate(R.layout.fragment_lista_libros,
                container, false);
        aa = new MiArrayAdapter(getActivity(), libros);
        ListView lv = (ListView) result.findViewById(R.id.listView);
        lv.setAdapter(aa);
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> parent, View view,
                        int position, long id) {
                                Libro l = aa.getItem(position);
                                //¡ PENDIENTE!
                        }
                });
        lv.setEmptyView(result.findViewById(R.id.emptyListView));
        return result;
} // onCreateView
```

Con todos estos cambios hemos conseguido que lo que antes eran actividades, ahora sean fragmentos. Lo que nos falta es tener una actividad que tenga ambos fragmentos.

1. Crea una nueva actividad. Vuelve a utilizar el nombre MainActivity que hemos hecho desaparecer al convertirla en un fragment. Asegúrate de que la marcas como actividad para el lanzador (Launcher activity).
2. Elimina las cosas habituales que nos mete el asistente: métodos de gestión del menú, menu_main.xml y las dos cadenas.
3. Comprueba que hemos vuelto a tener una actividad en el fichero de manifiesto.

4. Modifica el layout de la actividad para que incluya los dos fragmentos. Utiliza un LinearLayout en horizontal como hicimos antes, ajustando los pesos.
5. Elimina los android:padding_* de los layout de los dos fragmentos, para que no quede tanto espacio. Los incluyó el asistente cuando hizo los layouts pensando en que fueran usados en una actividad; al ser fragmentos, el espacio debe ser responsabilidad de la actividad que lo incluya. De hecho, en el layout de la actividad principal añadiremos margen entre los dos fragmentos.

```
<LinearLayout
... >
<fragment
android:id="@+id/libroFragment"
android:layout_width="0dp" android:layout_weight="1"
android:layout_height="match_parent"
android:layout_marginRight="12dp"
android:name="libro.azul.mislibros.ListaLibrosFragment">
</fragment>
<fragment
android:id="@+id/resumenFragment"
android:layout_width="0dp" android:layout_weight="2"
android:layout_height="match_parent"
android:name="libro.azul.mislibros.ResumenLibroFragment">
</fragment>
</LinearLayout>
```

6. Ejecuta la práctica. Comprueba que al seleccionar los diferentes elementos de la lista el fragmento del resumen no se actualiza.
7. Pulsa sobre alguno de los botones de la ordenación. Verás que la aplicación acaba con error.

La detección de la pulsación de los botones estaba hecha directamente en el layout usando android:onClick. Android busca los métodos indicados en el XML en la actividad en la que está el botón. Sin embargo, esos métodos los tenemos en el fragmento.

Podríamos moverlos a la actividad pero no es buena idea. Los fragmentos se utilizan para encapsular en una clase (y un layout) una porción del interfaz; es preferible que toda la lógica del fragmento la gestione él mismo.
La única forma de controlar los eventos de los botones en un fragmento es capturando el evento explícitamente. Es decir, tendremos que registrar un listener a mano por código para que llame a nuestros métodos.

1. Abre fragment_lista_libros y quita los atributos onClick de los dos botones, y añadele android:id para poder referirnos a ellos desde código (por ejemplo @+id/botonPorTituloy @+id/botonPorAutor.
2. Para capturar los eventos, añade en el onCreateView() el código de registro en los listener :

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

```
                onPorTitulo(v);
        }
});
        b = (Button) result.findViewById(R.id.botonPorAutor);
        b.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                onPorAutor(v);
        }
});

return result;

}//onCreateView
```

Fíjate que ese es el código que nos había ahorrado Android hasta ahora al permitir poner el nombre del método en el layout.

Ya sólo nos falta que al pulsar sobre un elemento de la lista se actualice el resumen. La detección de la pulsación la detectamos en el fragmento, y lo que queremos es modificar el fragmento "hermano". Podríamos, desde el fragmento de la lista, acceder a la actividad, y buscar en él el fragmento del resumen para manipularlo. Pero eso añade una dependencia entre los fragmentos, lo que dificulta la reutilización.

El esquema habitual es que el propio fragmento tenga un listener al que avise de que se ha seleccionado un elemento nuevo, y que la actividad se registre en él, y sea la actividad la que decida qué hacer.

De hecho, normalmente el fragmento asume que la actividad que lo está incluyendo implementa el interfaz y la registra como oyente automáticamente.

1. Añade en ListaLibrosFragment un interfaz interno público.

```
public interface OnLibroSeleccionadoListener {
        public void onLibroSeleccionado(Libro l);
}
```

2. Añade en ListaLibrosFragment un atributo nuevo que guarde un objeto que implemente ese interfaz. Llámalo _listener.

```
private OnLibroSeleccionadoListener _listener;
```

3. Haz que la actividad principal lo implemente. Deja de momento el código vacío.

```
public class MainActivity extends ActionBarActivity implements
        ListaLibrosFragment.OnLibroSeleccionadoListener {
// [ ... onCreate ... ]

        @Override
        public void onLibroSeleccionado(Libro l) {
        }
```

```
}
```

4. En el código del evento de la pulsación de un elemento de la lista que dejamos pendiente en el onCreateView, mira si el listener no es null, y llama al método del oyente en ese caso.

```
lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
                Libro l = aa.getItem(position);
                if (_listener != null)
                _listener.onLibroSeleccionado(l);
        }
});
```

Ahora lo que tenemos que hacer es que la actividad se registre para que reciba las notificaciones. En lugar de meter métodos para registro y desregistro y que la actividad se registre manualmente, aprovechamos dos métodos del ciclo de vida de los fragments y registramos a la actividad automáticamente. Añade al fragmento el siguiente código:

```
@Override
public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
                _listener = (OnLibroSeleccionadoListener) activity;
        } catch (ClassCastException e) {
                throw new ClassCastException(activity.toString()
                        + " debe implementar OnLibroSeleccionadoListener");
        }
}
@Override
public void onDetach() {
        super.onDetach();
        _listener = null;
}
```

```
/*Cambiar Activity por Context*/
public void onAttach(Context context) {
super.onAttach(context);
try {
_listener = (OnLibroSeleccionadoListener) context;
} catch (ClassCastException e) {
throw new ClassCastException(context.toString()
+ " debe implementar OnLibroSeleccionadoListener");
}
}
```

- El método onAttach() es llamado automáticamente cuando un fragmento se engancha en una actividad. Lo que hacemos es coger esa actividad y "registrarla" automáticamente como nuestro oyente. Si la actividad no implementa el interfaz, generamos una excepción que detendrá la aplicación. Esto significa que nuestro fragmento

exige que se implemente el interfaz siempre (¿para qué si no nos han incluído?)

- El método onDetach() es llamado cuando un fragmento se va a desacoplar de la actividad que lo contiene. Aprovechamos para "desregistrar" a la actividad como oyente. Fíjate que como un fragmento sólo puede estar conectado con una actividad no necesitamos una lista de oyentes.

Ya sólo falta reaccionar ante el evento. En la clase MainActivity mete un código similar al que ya usamos antes, pidiendo al gestor de fragmentos que nos dé el del resumen, y llamando a su método setLibro().

```
@Override
public void onLibroSeleccionado(Libro l) {
        ResumenLibroFragment rlf;
        rlf = (ResumenLibroFragment) getFragmentManager().
        findFragmentById(R.id.resumenFragment);
        rlf.setLibro(l.titulo, l.autor, l.resumen);
}
```

Lanza la aplicación, y comprueba que funciona. No obstante, al lanzar la aplicación en el fragmento de los detalles se ven los place holders.

1. Modifica el layout (fragment_resumen_libro) del fragmento de los detalles para que el nodo raíz sea un FrameLayout que tenga dentro al LinearLayout actual.
2. Añade, fuera del LinearLayout, un TextView centrado en la ventana, con el texto "No has seleccionado ningún libro"
3. Pon en el LinearLayout la visibilidad en gone.

```
<FrameLayout 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"
   tools:context="libro.azul.mislibros.ResumenLibroFragment">

   <TextView
      android:id="@+id/tvLibroNoSeleccionado"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center"
      android:text="@string/libroNoSeleccionado"
      style="@style/Base.TextAppearance.AppCompat.Headline"
      />

   <LinearLayout
      android:id="@+id/panelLibro"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:visibility="gone">

         <TextView…
         <TextView…
         <TextView…
```

```
</LinearLayout>
</FrameLayout>
```

4. Modifica el método setLibro() de ResumenLibroFragment para que se asegure de hacer visible el LinearLayout e invisible el TextView.

```
public void setLibro(String titulo, String autor, String resumen) {
        setVisibility(R.id.tvLibroNoSeleccionado, View.GONE);
        setVisibility(R.id.panelLibro, View.VISIBLE);
        setLabel(R.id.tituloLibro, titulo);
        ...
}
protected void setVisibility(int id, int visibility) {
        getView().findViewById(id).setVisibility(visibility);
}
```

Práctica 9.4: Fragmentos en actividades diferentes

Si la práctica anterior la ejecutas en un dispositivo con una pantalla pequeña, sufrirás evidentes problemas de espacio. Para resolverlo lo que querríamos es que se mostrara únicamente la lista de libros, y al pulsar sobre uno de ellos se mostrara su resumen. Vamos a modificar la práctica anterior para conseguirlo.

1. Haz una copia de la práctica anterior.
2. Crea un nuevo fichero de recurso (New/Layout Resource File), activity_main. Ponle el modificador de tamaño (Size) en Large.
3. Observa que ahora tendrás dos versiones del mismo fichero, uno de ellos marcado como large/.
4. Copia el contenido del activity_main anterior a la nueva versión.
5. Modifica la versión predefinida (para pantallas "pequeñas" y _normales") y quita el segundo fragment. Dado que ahora sólo tenemos un elemento, modifica el LinearLayout para que sea un FrameLayout. Revisa el fragment para poner match_parent en el ancho (y no usar el peso) y quita el margen.
6. Lanza un AVD pequeño y prueba la práctica sobre él. Observa que se utiliza el nuevo layout. Si pulsas sobre cualquier elemento la aplicación fallará. ¿Por qué?
7. Crea una actividad nueva. Llámala ResumenActivity.
8. Modifica el layout de la nueva actividad para que sea similar a la de la actividad principal, pero utilizando el fragmento del resumen. Puedes copiar la definición del fragmento de la versión large del layout. Además, de nuevo por eficiencia, utiliza un FrameLayout (no pongas 0dp en el ancho, y no pongas peso en el fragmento).
9. En la actividad principal, cuando se pulse sobre cualquier elemento tenemos ahora que decidir qué hacer. Si el layout que se ha cargado tiene el fragmento del resumen, lo utilizaremos para mostrarlo. Si no, necesitaremos lanzar la actividad secundaria. El código para hacerlo será equivalente al que utilizamos en la práctica 9.1.

```
@Override
public void onLibroSeleccionado(Libro l) {
ResumenLibroFragment rlf;
rlf = (ResumenLibroFragment) getFragmentManager().
```

```
findFragmentById(R.id.resumenFragment);
if (rlf != null)
// Estamos en una disposición con los dos fragmentos.
// Actualizamos el contenido.
rlf.setLibro(l.titulo, l.autor, l.resumen);
else {
// Tenemos que lanzar la segunda actividad.
Intent intent = new Intent(getApplicationContext(),
ResumenActivity.class);
intent.putExtra(ResumenLibroFragment.TITULO, l.titulo);
intent.putExtra(ResumenLibroFragment.AUTOR, l.autor);
intent.putExtra(ResumenLibroFragment.RESUMEN, l.resumen);
startActivity(intent);
}
} // onLibroSeleccionado
```

10. Lanza la aplicación en el móvil. Al elegir ahora un elemento, se abre la actividad secundaria con el fragmento de los detalles. Por el momento, sin embargo, no aparece ninguno.
11. En el onCreate() de la segunda actividad, recupera los datos extra del intent, y utiliza un código similar al de la actividad principal para obtener el fragment y llamar a su método setLibro() con los datos.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resumen);
        ResumenLibroFragment rlf;
        rlf = (ResumenLibroFragment) getFragmentManager().
        findFragmentById(R.id.resumenFragment);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
                rlf.setLibro(
                extras.getString(ResumenLibroFragment.TITULO),
                extras.getString(ResumenLibroFragment.AUTOR),
                extras.getString(ResumenLibroFragment.RESUMEN));
        } // if hay extras
} // onCreate
```

Práctica 9.5: Fragmentos: ciclo de vida

Los fragmentos también tienen su propio ciclo de vida, gestionado por la actividad. No es Android el que se encarga de llamarla, sino que las actividades se enriquecieron (con el gestor de fragmentos) para ir informando a los fragmentos de los eventos importantes.

Los fragmentos tienen los mismos métodos del ciclo de vida que las actividades salvo onRestart() y onRestoreInstanceState() que pasa a llamarse onViewStateRestored(). Se añaden además métodos nuevos, algunos de los cuales ya hemos visto.

1. Haz una copia de la práctica anterior.
2. Crea una nueva clase Java y llámala SpyFragment. Haz que herede de android.support.v4.app.Fragment.

3. Para evitar escribir mucho, copia el código de todos los métodos de la clase SpyActivity, dado que muchos se comparten. Copia también la definición del atributo TAG.
4. Convierte a públicos todos los métodos que acabas de copiar.
5. Borra el método onRestart(), que no existe en los fragmentos.

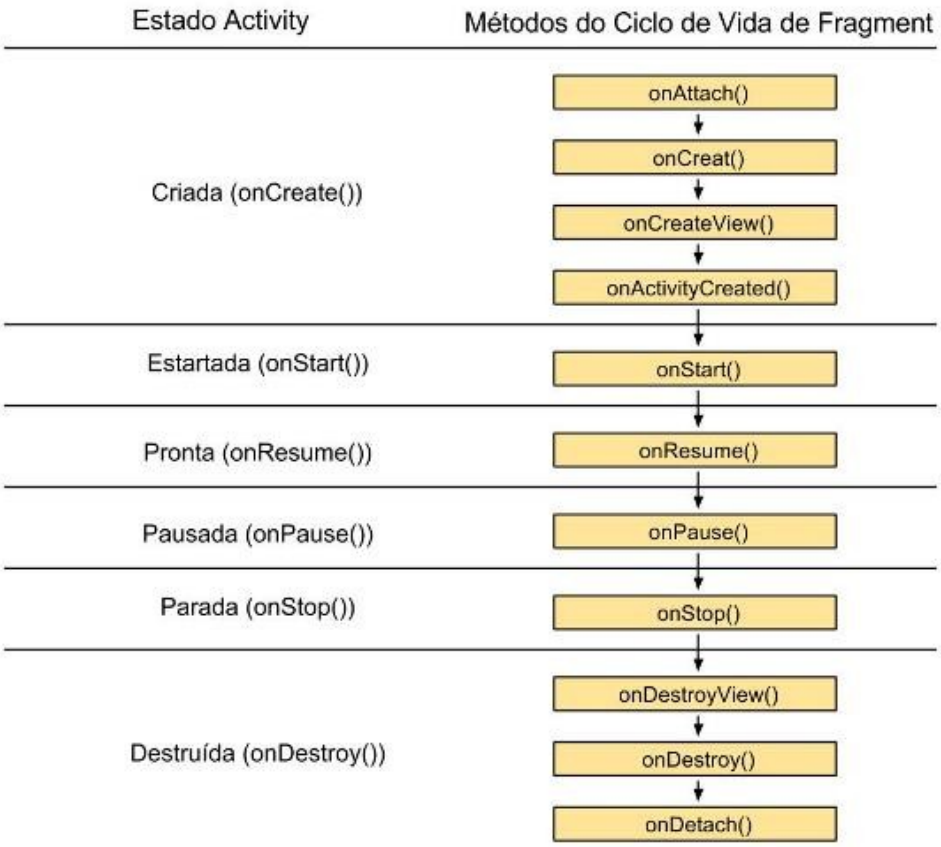

Figura 2.3: Ciclo de vida de un fragment

6. Renombra el método onRestoreInstanceState para que tenga el nombre usado en los fragmentos, onViewStateRestored, con el mismo parámetro. Ajusta el mensaje de log convenientemente.
7. Sobreescribe los métodos específicos del ciclo de vida de los fragmentos. Crea además un constructor por defecto y escribe un mensaje en log.

```
public SpyFragment() {
        android.util.Log.i(TAG, "<constructor>");
}
@Override
public void onAttach(Activity activity) {
        android.util.Log.i(TAG, "onAttach(" + activity + ")");
```

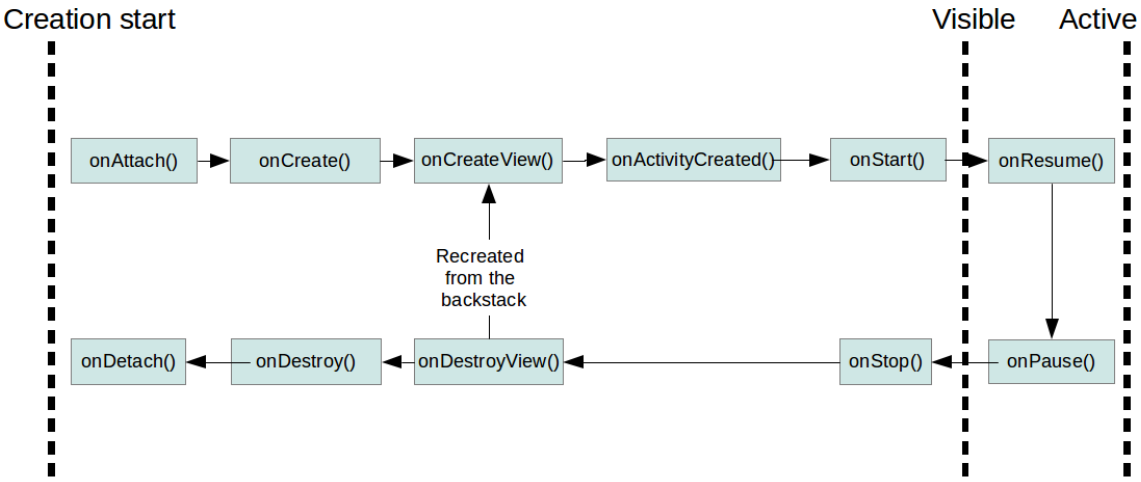

Figura 2.4: Emparejamiento de cada evento del ciclo de vida

```
super.onAttach(activity);
}
@Nullable
@Override
public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState) {
        android.util.Log.i(TAG, "onCreateView()");
        return super.onCreateView(inflater,
        container, savedInstanceState);
}
@Override
public void onActivityCreated(
                        @Nullable Bundle savedInstanceState) {
        if (savedInstanceState == null)
                android.util.Log.i(TAG, "onActivityCreated()");
        else
                android.util.Log.i(TAG, "onActivityCreated(" +
                                                savedInstanceState + ")");
        super.onActivityCreated(savedInstanceState);
}
@Override
public void onDestroyView() {
        android.util.Log.i(TAG, "onDestroyView()");
        super.onDestroyView();
}
@Override
public void onDetach() {
        android.util.Log.i(TAG, "onDetach()");
        super.onDetach();
}
```

8. Modifica los dos fragmentos para que hereden de nuestra nueva clase. Modifica también la actividad principal para que herede de SpyActivity, si no lo hacía ya.
9. Ejecuta la aplicación y ciérrala. Observa cómo los eventos principales ocurren primero en la actividad e inmediatamente después en los

fragmentos. Además, algunas veces los eventos adicionales de un mismo fragmento van juntos y otras veces no.

Práctica 9.6: Configuración diferente en horizontal y vertical

En la práctica 9.4 conseguimos tener dos fragmentos en dispositivos de tamaño grande, y dos actividades en dispositivos más pequeños. Sin embargo, podríamos querer mostrar los dos fragmentos en móviles apaisados, y dejar el funcionamiento de dos actividades únicamente para móviles en vertical (retrato).

Para eso, tendremos que revisar nuestros layouts. Ahora mismo tenemos como layout predefinido el que muestra un único fragmento, que sobreescribimos para pantallas grandes con la versión de dos. Lo que queremos es que esa misma versión se utilice también en pantallas pequeñas en apaisado.

Podríamos cambiar el esquema y poner como predefinido la versión con dos fragmentos, y sobreescribirlo con la versión de uno en móviles pequeños en modo retrato.

En lugar de hacer eso, vamos a aprovechar para ver cómo utilizar un mismo recurso en dos configuraciones diferentes a la predefinida usando un alias para evitar tener el mismo fichero dos veces.

1. Haz una copia de la práctica anterior.
2. Entra en el directorio layout-large de los recursos, y renombra el fichero activity_main.xml por activity_main_twopanes.xml y muevelo al directorio layout. Ten en cuenta que cualquier nombre servirá.
3. Crea un nuevo activity_main para los dispositivos de pantalla grande, y crea un alias al layout que acabas de renombrar:

```
<?xml version="1.0" encoding="utf-8"?>
<merge>
<include layout="@layout/activity_main_twopanes"/>
</merge>
```

Ha dejado de funcionar include dentro de merge.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include
        layout="@layout/activity_main_twopanes"/>
</LinearLayout>
```

4. Crea un directorio layout-land y copia el mismo fichero con el alias.
5. Lanza la aplicación en un móvil, y comprueba que al cambiar la orientación, se pasa de una representación a otra.

6. Observa los mensajes del ciclo de vida. Dependiendo de cuántos fragmentos haya en la orientación destino, se pasa por el ciclo en el de la lista únicamente, o en los dos.
7. Cierra la aplicación en el móvil, y vuelve a lanzarla con el móvil en vertical (modo retrato).
8. Pulsa sobre cualquier libro, y observa que salta la actividad correctamente.
9. Pulsa “Volver” para volver a la actividad principal, y rota el móvil. Verás la versión con los dos fragmentos.
10. Pulsa sobre cualquier libro y verás su resumen.
11. Rota el móvil, y volverás a la lista.
12. Pulsa sobre cualquier libro. ¿Qué ocurre y por qué?
13. Los fragmentos que se han creado en algún momento se recrean automáticamente, aunque luego puede que no se asocien a ningún lugar de la disposición. En la actividad principal, al seleccionar un libro mirábamos si existía el fragmento del resumen a partir de su identificador, y en ese caso establecíamos el libro. Ahora el fragmento existe, y con ese identificador, pero no tiene la vista creada (no se ha lanzado su ciclo de vida), por lo que falla. Tenemos que mejorar esa condición:

```
public void onLibroSeleccionado(Libro l) {
        // ...
        if ((rlf != null) && rlf.isInLayout())
        rlf.setLibro(l.titulo, l.autor, l.resumen);
        else {
                ...
        }
}
```

14. Lanza de nuevo la aplicación y comprueba que ahora funciona bien.
15. Con el móvil en modo retrato, selecciona un libro para ver su resumen.
16. Gira el móvil para verlo en apaisado. ¿Es razonable lo que ves? Pulsa “Volver”. ¿Qué ocurre ahora? ¿Cómo podemos evitarlo?
17. La actividad que muestra el resumen no queremos que sea visible nunca en modo "apaisado". La solución es ajustar el método onCreate() para que si detecta durante el onCreate() que la configuración es apaisada, se finalice a sí misma.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_LANDSCAPE) {
                finish();
                return;
        }
        setContentView(R.layout.activity_resumen);
        ...
} // onCreate
```

Práctica 9.7: Dos fragmentos dinámicos

En las prácticas anteriores hemos hecho un uso estático de los fragments, dado que en los ficheros de layout está integrado directamente la posición de cada uno, y la clase con la lógica que hay que instanciar.

Una aproximación diferente es hacer uso de fragmentos dinámicos. En los layout colocamos ViewGroup's como contenedores vacíos, sobre los que desplegaremos los fragmentos en ejecución.

Vamos a empezar otra vez, construyendo la aplicación, asumiendo que sólo queremos que se utilice en pantallas grandes donde mostraremos los dos fragmentos, tal y como hicimos inicialmente en la práctica 9.3.

1. Haz una copia de la práctica anterior.
2. Renombra el paquete principal por DM2E.mislibrosdinamicos, dado que vamos a hacer cambios drásticos en la filosofía de funcionamiento. Modifica la cadena app_name para hacer más explícita la diferencia y poder distinguir ambas versiones instaladas en el dispositivo. Pon, por ejemplo, Mis libros (fragments dinámicos).
3. Dado que vamos a asumir siempre que tenemos visibles los dos paneles, elimina los alias del layout de pantallas grandes y apaisadas. Elimina la versión predefinida con un solo panel, y renombra el fichero activity_main_twopanes.xml para que sea activity_main.xml.
4. Modifica el layout para convertir los elementos <fragment>en elementos <FrameLayout>que serán nuestros contenedores de los fragmentos. Elimina en ambos el atributo android:name que ya no se utiliza y mueve el marginRight del panel izquierdo al derecho, convirtiéndolo en un marginLeft.
5. Ejecuta la aplicación. Comprueba que no aparece nada.

Dado que no hemos metido los fragments en el layout, tenemos que meterlos por código en el método onCreate(). La gestión de los fragments de la actividad la realiza la clase FragmentManager, que ya hemos utilizado en algún momento para buscar fragmentos en la actividad. Si queremos añadir, eliminar o sustituir fragmentos, tendremos que hacer uso también de esa clase.

La modificación de los fragmentos se realiza a través de transacciones. Para añadir un nuevo fragmento, le diremos al FragmentManager que queremos iniciar una transacción, luego indicaremos los cambios que queremos hacer, y finalmente indicaremos que hemos terminado y deben ejecutarse nuestros cambios.

1. En el código del método onCreate() de la actividad principal, pide al gestor de los fragmentos una nueva transacción, indica que deseas añadir un fragmento de tipo lista en el panel izquierdo, y consolida la transacción.

```
ListaLibrosFragment llf = new ListaLibrosFragment();
FragmentTransaction transaction;
transaction = getSupportFragmentManager().beginTransaction();
transaction.add(R.id.contenedorListaLibros, llf);
transaction.commit();
```

2. Ejecuta la aplicación y comprueba que aparece nuestro fragment. Pulsa “Volver” para cerrarla.

3. El uso de transacciones permite al sistema mantener una pila de vuelta, de modo que le podemos pedir que recuerde cada transacción que ha realizado, y la deshaga cuando se pulse el botón “Volver”. Aunque en principio en este momento no tenga ningún sentido dentro de la aplicación, antes de consolidar la transacción llamando a commit pide al sistema que añada la transacción actual a la pila y la recuerde:

```
transaction.addToBackStack(null);
```

1. Ejecuta la aplicación y comprueba que no parece haber cambiado nada.
2. Pulsa el botón “Volver”. Observa que el fragmento desaparece al deshacerse la última transacción.
3. Vuelve a pulsar “Volver”. La aplicación se cierra.
4. Lanza de nuevo la aplicación. Rota el dispositivo.
5. Pulsa “Volver”. No parecerá ocurrir nada. Pulsa una segunda vez y observa que, ahora sí, el fragmento desaparece.
6. Relanza la aplicación, rota el dispositivo dos veces, e intenta cerrar el programa pulsando “Volver”. ¿Qué está ocurriendo?
7. Lo que ocurre es que el gestor de fragmentos conserva el estado entre reinicios de la actividad. Al rotarlo una vez, la actividad se cierra y reconstruye, y el gestor de fragmentos nos añade automáticamente el fragmento que añadimos inicialmente. Nosotros en el onCreate() añadimos siempre el fragmento, lo que significa que se añadirá dos veces (o más, si has rotado varias veces el dispositivo). Modifica el onCreate() para añadir el fragmento sólo si es la primera ejecución:

```
protected void onCreate(Bundle savedInstanceState) {
...
if (savedInstanceState == null) {
ListaLibrosFragment ...
...
}
} // onCreate
```

8. Vuelve a lanzar la aplicación y comprueba que ya no se acumulan múltiples copias del fragmento.
9. No tiene ningún sentido el comportamiento actual con el botón "Volver". Quita la llamada a addToBackStack(null).
10. Vamos a convertir la “vista” que avisaba de que no se había elegido ningún libro en un fragment. Para eso, lo primero es independizarlo. Crea un layout nuevo y llámalo fragment_sinlibroseleccionado.xml.
11. Copia el TextView original del layout fragment_resumen_libro. No hace falta que dejes el layout. Podemos poner directamente el TextView como raíz.
12. Dado que ya no tenemos el mensaje en el layout del fragmento del resumen, elimina el FrameLayout que metimos antes y deja el LinearLayout como nodo raíz. Quita también el android:visibility="gone". Por último, modifica la clase

ResumenLibroFragment para que no se manipule la visibilidad de los dos elementos en el método setLibro().

13. Crea una nueva clase de Java (no uses el asistente de fragmentos) y llámala SinLibroSeleccionadoFragment.
14. Haz que herede de Fragment.
15. Implementa el método onCreateView(...), que expanda el layout anterior:

```
public class SinLibroSeleccionadoFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater,
                                        ViewGroup container,
                                        Bundle savedInstanceState) {
                return inflater.inflate(
                        R.layout.fragment_sinlibroseleccionado,
                        container, false);
        }
} // SinLibroSeleccionadoFragment
```

16. Modifica el onCreate de la actividad principal para que en la transacción que ya hay se añada también el fragmento en el segundo contenedor:

```
if (savedInstanceState == null) {
        ListaLibrosFragment llf = new ListaLibrosFragment();
        SinLibroSeleccionadoFragment slsf =
                                new SinLibroSeleccionadoFragment();
        FragmentTransaction transaction;
        transaction = getSupportFragmentManager().beginTransaction();
        transaction.add(R.id.contenedorListaLibros, llf);
        transaction.add(R.id.contenedorResumenLibros, slsf);
        transaction.commit();
}
```

17. Para que al pulsar cualquier elemento de la lista se muestren sus detalles, tenemos que modificar el método onLibroSeleccionado(), construyendo un fragmento y realizando una transacción con él. Ahora no queremos añadir el fragmento, sino que queremos reemplazar el que ya hay con el nuevo.

```
public void onLibroSeleccionado(Libro l) {
        ResumenLibroFragment rlf;
        rlf = new ResumenLibroFragment();
        rlf.setLibro(l.titulo, l.autor, l.resumen);
        FragmentTransaction transaction;
        transaction = getSupportFragmentManager().beginTransaction();
        transaction.replace(R.id.contenedorResumenLibros, rlf);
        transaction.addToBackStack(null);
        transaction.commit();
}
```

18. Lanza la aplicación y pulsa sobre algún libro. ¿Qué ocurre?

La aplicación falla porque hemos llamado a setLibro() antes de que el fragmento se inicialice, y no es capaz de encontrar los elementos de la

vista. Lo que tenemos que hacer es conseguir que el fragmento guarde las cadenas temporalmente, y cuando expanda la vista en el onCreateView() las utilice para establecerlas en las etiquetas.

Hay un problema adicional y es que si el dispositivo se rota, hemos visto que es Android quién se encarga de reconstruir nuestros fragmentos; en el onCreate() de la actividad no estamos haciendo nada con los fragmentos si nos estamos reconstruyendo, porque Android lo hará por nosotros.

Pero Android llama al constructor por defecto de los fragmentos pero no llamará al método setLibro() para que el objeto recupere las cadenas. Podríamos capturar los eventos del ciclo de vida del fragmento para conservar su estado, y luego usarlo para reconstruirlo, pero hay una opción mejor.

Dado que el problema de la inicialización de los fragmentos por código como estamos haciendo aquí es similar al problema de la reconstrucción de los fragmentos de manera automática, Android añade en los fragmentos un bundle adicional, que se guarda automáticamente sin que tengamos que preocuparnos nosotros en los métodos del ciclo de vida. Es decir, cualquier cosa que guardemos en el bundle obtenido con getArguments() de un fragmento sobrevivirá a cambios de configuración de manera automática.

El esquema de programación de los fragmentos lo que hace es aprovechar esto también para la inicialización de los fragmentos por código. En concreto, en lugar de tener nuestro setLibro() que recibe los parámetros y los guarda, tal y como proponíamos en sus atributos, lo que hacemos es guardarlos en el bundle "de guardado automático", para saber que los tendremos ahí la próxima vez que nos reconstruyan. En el onCreateView() sacamos siempre los parámetros de ese bundle, y así no tenemos que diferenciar si los cogemos de nuestros supuestos atributos de "construcción en dos pasos", o del bundle recibido como parámetro en el onCreateView().

Además, la construcción habitual es crear un método factoría (estático) llamado newInstance() que reciba los parámetros de inicialización y que haga todo el trabajo por nosotros.

Modifica la clase ResumenLibroFragment para que quede algo así:

```
public class ResumenLibroFragment extends SpyFragment {
public static final String TITULO = "titulo";
public static final String AUTOR = "autor";
public static final String RESUMEN = "resumen";
public static ResumenLibroFragment newInstance(String titulo,
String autor,
String resumen) {
ResumenLibroFragment fragment = new ResumenLibroFragment();
Bundle args = new Bundle();
args.putString(TITULO, titulo);
args.putString(AUTOR, autor);
args.putString(RESUMEN, resumen);
fragment.setArguments(args);
return fragment;
}
```

```
public ResumenLibroFragment() {
}
@Override
public View onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle savedInstanceState) {
View result = inflater.inflate(R.layout.fragment_resumen_libro,
container, false);
if (getArguments() != null) {
Bundle args = getArguments();
setLabel(result, R.id.tituloLibro, args.getString(TITULO));
setLabel(result, R.id.autorLibro, args.getString(AUTOR));
setLabel(result, R.id.resumenLibro, args.getString(RESUMEN));
}
return result;
}
protected void setLabel(int id, String str) {
setLabel(getView(), id, str);
} // setLabel
protected void setLabel(View v, int id, String str) {
TextView tv;
tv = (TextView) v.findViewById(id);
if (str != null)
tv.setText(str);
else
tv.setText("");
} // setLabel
} // ResumenLibroFragment
```

1. Ejecuta el programa.
2. Selecciona algún libro.
3. Rota el dispositivo y comprueba que aunque la actividad se ha reconstruido, se sigue viendo el contenido.
4. Selecciona varios libros, y vuelve a rotar el dispositivo.
5. Ve pulsando “Volver” y comprueba que todo funciona como debería.
6. Antes de terminar, elimina la llamada a addToBackStack(null) en MainActivity para que no se conserven las transacciones. Es antinatural que el botón volver tenga este significado aquí.

Práctica 9.8: Fragmentos estáticos estables

Vamos a recuperar la última versión de la práctica usando fragments estáticos para utilizar en el fragmento con el resumen del libro el mismo esquema que hemos utilizado en la práctica anterior. De ese modo, conseguiremos que se mantenga el resumen del último libro cuando rotemos el dispositivo.

1. Haz una copia de la práctica 9.6.
2. Ejecútala sobre un dispositivo con pantalla grande. Selecciona un libro y rota la pantalla. Comprueba que el resumen se pierde. Es debido a que la invocación a setLibro() que se hace desde el evento de la actividad principal hace cambios no persistentes en el fragmento, y no nos hemos preocupado de grabarlo.

3. En lugar de grabarlo usando el ciclo de vida del fragmento, nos aprovecharemos de los argumentos del fragmento. Copia la nueva versión de setLabel() que hemos puesto en la práctica anterior para que reciba la vista en la que buscar el TextView en lugar de hacerlo en la vista del fragmento.
4. Modifica onCreateView() para que, si getArguments() no es null, extraiga del bundle los campos del título, autor y resumen y los establezca en las etiquetas correspondientes, tal y como hicimos antes. Recuerda, además, hacer visible el LinearLayout y ocultar el TextView con el aviso, tal y como tenemos en setLibro().

    - Añadimos al principio de

```
if (getArguments() != null) {
setVisibility(result, R.id.tvLibroNoSeleccionado, View.GONE);
setVisibility(result, R.id.panelLibro, View.VISIBLE);
...
```

    - Hay que modificar el método setVisibility y añadir un argumento más

```
View v.
```

    - Cambiamos en setLibros

```
setVisibility(getView(), R.id.tvLibroNoSeleccionado, View.GONE);
setVisibility(getView(), R.id.panelLibro, View.VISIBLE);
```

5. Modifica el método setLibro() para que, además de cambiar las etiquetas, modifique el getArguments() estableciendo los nuevos valores en los parámetros. Así garantizaremos que se conservarán entre reinicios.

```
public void setLibro(String titulo, String autor, String resumen) {
...
Bundle args = getArguments();
args.putString(TITULO, titulo);
args.putString(AUTOR, autor);
args.putString(RESUMEN, resumen);
} // setLibro
```

6. Ejecuta la aplicación. Fallará, porque getArguments() es null.
7. No podemos establecer un bundle como parámetro del fragmento una vez que éste está ya en marcha, por lo que no podemos hacer el new directamente ahí. En el constructor, añade:

```
setArguments(new Bundle());
```

8. Vuelve a ejecutar la aplicación. Comprueba que ahora ya sí funciona.

Notas bibliográficas

http://developer.android.com/training/basics/fragments/index.html

http://developer.android.com/guide/components/fragments.html
http://developer.android.com/reference/android/app/Fragment.html
http://developer.android.com/guide/topics/resources/accessing-resources.html