Capítulo 14
Animaciones

Resumen: En este capítulo veremos el sistema de animaciones integrado en Android desde su primera versión. Es útil para realizar efectos dinámicos sencillos que, además, se pueden especificar como recursos (en XML), lo que facilita su reutilización.

Práctica 7.1: Animaciones básicas entre fragmentos

Hoy en día el usuario espera interfaces dinámicas y amigables. Las animaciones de elementos del interfaz de usuario se han convertido en habituales en cualquier aplicación, y Android proporciona soporte para ellas de diferentes formas. En esta práctica vamos a ver las animaciones que soporta Android en las transacciones entre fragmentos.

1. Haz una copia de la práctica 2.7. En ella, mostrábamos en una actividad, dos fragmentos dinámicos, uno con la lista de libros y otro con el resumen del libro seleccionado. Este último fragmento lo sustituimos bajo demanda.
2. Busca el código donde se hacía el cambio de fragmento. Estaba en la actividad principal, en el método onLibroSeleccionado().
3. Una vez abierta la transacción entre fragmentos, con gura la transición (animación) a reproducir.

transaction = getSupportFragmentManager().beginTransaction(); transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_*);

...

transaction.commit();

4. Los posibles valores para el parámetro son TRANSIT_FRAGMENT_NONE, TRANSIT_FRAGMENT_OPEN y TRANSIT_FRAGMENT_CLOSE. Prueba los tres. Es preferible que utilices un dispositivo físico.

Práctica 7.2: Animaciones personalizadas entre fragmentos

Las posibilidades de las transiciones predeterminadas son muy escasas. Afortunadamente, es posible seleccionar animaciones independientes para cada uno de los dos fragmentos que entran en juego en la transacción.

1. Haz una copia de la práctica anterior.
2. En lugar de llamar al método setTransition() para poner una transición prede nida, llama asetCustomAnimations(). Recibe dos parámetros, con la animación que quieres poner al fragmento entrante y al saliente. Por ejemplo:

transaction.setCustomAnimations(android.R.anim.slide_in_left,
android.R.anim.slide_out_right);

3. Lanza la práctica y comprueba el efecto.
4. Si no lo estaba ya, pide a la transacción que se guarde en la pila de vuelta:

transaction.addToBackStack(null);

5. Prueba la práctica de nuevo. Pulsa varios libros y pulsa volver. Al deshacer, no se ejecutan animaciones.
6. setCustomAnimations tiene una segunda alternativa con dos parámetros más para especificar la animación asignada al fragmento entrente y saliente al pulsar el botón Volver. Con gura las animaciones de volver con las mismas animaciones:

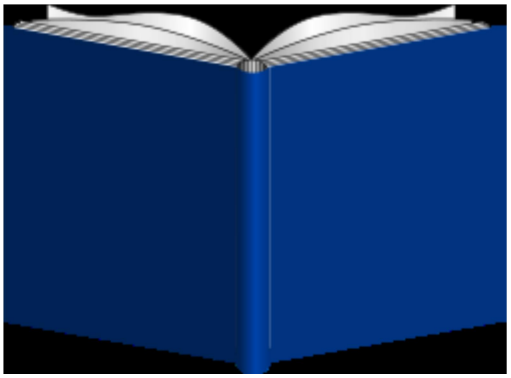
transaction.setCustomAnimations(android.R.anim.slide_in_left,
android.R.anim.slide_out_right, android.R.anim.slide_in_left, android.R.anim.slide_out_right);

7. Comprueba el efecto. ¿Qué te parece? ¿Cambiarías algo?

Práctica 7.3: Animaciones "tween": escalado

Las animaciones que acabamos de aplicar al fragmento son animaciones "tween" (como versión abreviada de inbetween), que son animaciones interpoladas sobre vistas. Están soportadas desde la primera versión de Android y, aunque en el nivel de API 11 fueron mejoradas por las animaciones de propiedades, aún siguen siendo muy utilizadas. Se especifican o bien por código o, mejor, a través de un XML que almacenaremos en el directorio res/anim, lo que permite especificar efectos visuales sencillos pero llamativos que resultan fáciles de reutilizar para dinamizar nuestro interfaz de usuario. Ésta será la primera de una secuencia de prácticas para experimentar con estas animaciones. Puedes realizar todas sobre el mismo proyecto, o bien ir haciendo copias del proyecto anterior e incluso cambiando el nombre del paquete para poder tenerlas todas instaladas en el móvil simultáneamente.

1. Crea un proyecto nuevo y llámalo DM2E.tween[1].
2. Mete en el directorio de recursos drawable una imagen cualquiera (por ejemplo, un libro azul). Procura que la imagen que escojas se vea bien en el dispositivo donde vayas a probarlo y que no sea simétrica ante giros.



3. Modifica el layout predefinido para quitar la etiqueta introducida por el asistente, y pon un ImageView que muestre la imagen en el centro de la actividad. Además, pon en el layout raíz un evento onClick para que se ejecute un método al pulsar sobre cualquier espacio vacío de la actividad.

<RelativeLayout ... android:onClick="onClickView">

<ImageView android:id="@+id/imagen"

[1] Si pretendes diferenciar cada práctica, puedes utilizar mejor el nombre DM2E.tweenscale.

```
android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:src="@drawable/libroazul"

android:layout_centerInParent="true"/>

</RelativeLayout>
```

4. En el directorio res crea un nuevo directorio anim.
5. Crea un chero de recurso de animación tween.xml en él.
6. La plantilla crea un nodo raíz de tipo <set>. De momento seremos menos ambiciosos y lo cambiaremos por <scale>. Modifica el contenido para que tenga una animación de escalado que empiece con escala 1 (tamaño original) y acabe con escala 0 para ver desvanecerse al libro.

```
<scale
xmlns:android="http://schemas.android.com/apk/res/android"
android:duration="1000"
android:fromXScale="1.0"
android:fromYScale="1.0"
android:toXScale="0.0"
android:toYScale="0.0"/>
```

7. Para ejecutar la animación, implementa el método llamado ante la pulsación en la actividad. Tendrás que buscar la vista del libro, y asociarle la animación:

```
public void onClickView(View v)
{
View imagen = findViewById(R.id.imagen);
Animation anim;
anim = AnimationUtils.loadAnimation(this, R.anim.tween);
 imagen.startAnimation(anim);
}
```

8. Ejecuta la práctica. ¿Qué conclusiones sacas?
9. Añade como atributo android:fillAfter="true". ¿Para qué sirve?
10. Añade como atributo android:pivotX="50%". ¿Para qué sirve?

Práctica 7.4: Animaciones “tween”: traslación

1. A partir de la práctica anterior (o de una copia), modifica la animación para hacer uso de translate en lugar de scale:

```
<translate xmlns:android="..."
android:duration="1000"
android:fillAfter="true"
android:fromXDelta="20%"
android:fromYDelta="20%"
android:toXDelta="100%"
android:toYDelta="0%"/>
```

2. Ejecuta la práctica. ¿Qué conclusiones sacas?

3. Modifica la animación anterior, pon 0% en los atributos from* y cambia toXDelta por 50%p. ¿Qué indica la p?
4. Modifica el layout y sustituye el paddingRight por layout_marginRight. Ejecuta la práctica. ¿Cómo explicas el resultado?
5. Deshaz el último cambio.

Práctica 7.5: Animaciones “tween”: rotación

1. A partir de la práctica anterior (o de una copia), modifica la animación para hacer uso de rotate en lugar de translate:

```
<rotate xmlns:android="..."
android:duration="1000"
android:fillAfter="true"
android:fromDegrees="0"
android:toDegrees="360"/>
```

2. Ejecuta la práctica. ¿Qué conclusiones sacas? ¿Cómo conseguirías que rotara “sobre sí mismo”?

Práctica 7.6: Animaciones “tween”: alfa

1. A partir de la práctica anterior (o de una copia), modifica la animación para hacer uso de alpha:

```
<alpha xmlns:android="..."
android:duration="1000"
android:fillAfter="true"
android:fromAlpha="1"
android:toAlpha="0" />
```

2. Ejecuta la práctica.

Práctica 7.7: Animaciones “tween”: animaciones simultáneas

1. Utilizando como nodo raíz <set>, es posible especificar múltiples animaciones que se ejecutarán todas a la vez. Modifica la práctica anterior (o una copia) para crear una animación que rote, y escale simultáneamente:

```
<set xmlns:android="..."
android:duration="1000"
android:fillAfter="true">

<scale android:fromXScale="100%"
android:fromYScale="100%"
android:toXScale="0%"
android:toYScale="0%"
android:pivotX="50%"
android:pivotY="50%"/>
```

```
<rotate android:fromDegrees="0"
android:toDegrees="360"
android:pivotX="50%"
android:pivotY="50%"/>

 </set>
```

2. Ejecuta la práctica.
3. Observa que hemos puesto la duración de manera global en el set. Quita la especi cación de la duración de set, y pon una duración de 1 segundo al escalado y de medio segundo a la rotación y observa el resultado.
4. Deshaz el último cambio.
5. Sustituye el escalado por una traslación:

```
<translate android:fromXDelta="0%"
android:fromYDelta="0%"
android:toXDelta="100%"
android:toYDelta="0%"/>
```

6. Ejecuta la práctica. ¿Hace lo que esperabas? ¿Por qué? Modifica la animación para que realice el efecto que estabas esperando.

Práctica 7.8: Animaciones “tween”: animaciones en secuencia

Las animaciones tienen un atributo startOffset para indicar en qué momento queremos que comience respecto al instante de inicio del bloque. Si se utiliza, el <set> padre no deberá tener tiempo global.

1. Modifica la práctica anterior (o una copia) para que el <set> no tenga tiempo global.
2. Pon una duración de un segundo a la animación de traslación.
3. Pon una duración de medio segundo a la rotación, y startOffset de 500.
4. Ejecuta la práctica.

Práctica 7.9: Animaciones “tween”: fillAfter y fillBefore

Con lo que sabes hasta ahora y usando como base la práctica anterior, ¿eres capaz de hacer una animación que haga desaparecer al libro (escala 0) y luego volverlo a hacer aparecer (escala 1)?

```
<set
        xmlns:android="http://schemas.android.com/apk/res/android">
<scale android:duration="500"
android:fromXScale="1"
android:fromYScale="1"
android:toXScale="0"
android:toYScale="0"
android:fillEnabled="true"
android:fillAfter="false"/>
```

```
<scale android:startOffset="500"
android:duration="500"
android:fromXScale="0"
android:fromYScale="0"
android:toXScale="1"
android:fillEnabled="true"
android:fillBefore="false"
android:toYScale="1"/>

</set>
```

Notas bibliográficas

http://developer.android.com/guide/topics/resources/animation-resource. html

http://developer.android.com/guide/topics/graphics/view-animation. html

http://developer.android.com/guide/topics/graphics/drawable-animation. html

Curso: IFC02CM15 - Programación avanzada de dispositivos móviles - Julio 2015 Pedro Pablo Gómez Martín