**Unidad 1 – Diseño avanzado de interfaces de usuario**

ENLACES POLIFORMAT UNIDAD 1

[Ud 1-Diseño avanzado de interfaces de usuario](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main)

|  |
| --- |
| [Introducción](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Acceder a objetos globales de la aplicación](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [La clase Application](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Uso del patrón Singleton](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [GridView](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Fragments](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Insertar fragments desde xml](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Insertar fragments desde código](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Comunicación e intercambio de fragments](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [La barra de acciones (ActionBar)](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [La barra de acciones en versiones anteriores a la 3.0](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Añadiendo pestañas en la barra de acciones](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Añadiendo preferencias de usuario mediante PreferenceFragment](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Servicios de búsquedas](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Navigation Drawer](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Animaciones](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Animaciones de vistas: Transiciones entre actividades](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Aplicando animaciones de vistas en Audiolibros](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Animaciones de propiedades](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [El motor básico de animación: ValueAnimator](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Automatizando las animaciones: ObjectAnimator](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Combinando animaciones: AnimatorSet](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Definiendo animaciones en XML](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Nuevas propiedades de la clase View](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |
| [Aplicando animaciones de propiedades en Audiolibros](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/main) |

# 1.1. Introducción

|  |  |
| --- | --- |
| |  | | --- | |  | |
| Las últimas versiones de Android incorporan nuevas herramientas para el diseño de la interfaz de usuario. A partir de la versión 3.0 se añaden nuevos mecanismos de interacción con el usuario, como la barra de acciones o los *fragments*. La mayoría de estos mecanismos han sido introducidos para permitir un diseño que se adapte a diferentes tamaños de pantalla. El objetivo es que nuestras aplicaciones se visualicen correctamente, tanto en pequeños teléfonos móviles, como en tabletas y televisores.  En esta unidad comenzaremos con un tema alejado de la línea general del capítulo: cómo almacenar información a la que se pueda acceder desde cualquier parte de nuestra aplicación. Describiremos dos alternativas, el uso de la clase Application y del patrón *Singleton*.  En el segundo apartado trataremos la vista GridView, que nos permite mostrar una cuadrícula de elementos deslizables verticalmente. Cada uno de los elementos de esta cuadrícula puede ser definido con una vista personalizada.  En el tercer apartado aprenderemos a usar *fragments*. Su utilización es fundamental, dado que el nuevo planteamiento de diseño de la interfaz de usuario en Android se basa en *fragments*. Se trata de elementos constructivos básicos que podremos combinar dentro del *layout* de una actividad. Aunque aparecen en la versión 3.0, pueden usarse en versiones anteriores gracias a una librería de compatibilidad.  En el cuarto apartado se introduce la barra de acciones o ActionBar que se muestra en la parte superior de las aplicaciones. A partir de la versión 3.0 este elemento viene a reemplazar el menú de la aplicación. También se describe cómo podemos incorporar un servicio de búsquedas y pestañas dentro de esta barra.  En el siguiente apartado incorporaremos un nuevo elemento a nuestra interfaz de usuario, el *Navigation Drawer*. Este mecanismo de navegación consiste en un menú lateral deslizable que podemos mostrar pulsando un botón de la barra de acciones.  En el último apartado veremos las alternativas para introducir animaciones. Comenzaremos repasando las animaciones de vistas y cómo aplicarlas para introducir transiciones entre actividades. Finalmente, se estudiarán con detalle las animaciones de propiedades. Una API que nos facilitará la realización de animaciones sobre cualquier tipo de objeto.  **Objetivos:**   Mostrar dos alternativas para acceder a información global de la aplicación.   Describir el uso de GridView para visualizar una cuadrícula de vistas.   Mostrar como usando *fragments* podemos diseñar elementos reutilizables de la IU.   Aprender a intercambiar dinámicamente los *fragments* mostrados en una actividad.   Describir el uso de la barra de acciones.   Implementar un servicio de búsquedas.   Enumerar los pasos a seguir para insertar un Navigation Drawer.   Repasar las alternativas para hacer animaciones en Android.   Mostrar cómo podemos hacer transiciones entre actividades mediante animaciones de vistas.   Describir la API para animaciones de propiedades. |

# 1.2. Acceder a objetos globales de la aplicación

|  |
| --- |
| Cada uno de los componentes de una aplicación se escribe en una clase separada. Esto hace que en muchas ocasiones resulte complicado compartir objetos entre estos componentes.  Para poder acceder a una información global desde cualquier clase de nuestro proyecto, podemos utilizar el modificador static. De esta forma, no será necesario conocer la referencia a un objeto de la clase, solo con indicar el nombre de la clase podremos acceder a esta información.  Otra alternativa, muy similar a la anterior, es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia a la que se podrá acceder desde cualquier sitio utilizando un método estático. Lo veremos más adelante.  Una tercera alternativa específica de Android consiste en crear un descendiente de la clase Application. En el siguiente punto se explica cómo hacerlo. |

## 1.2.1. La clase Application

|  |
| --- |
| Esta clase ha sido creada en Android para almacenar información global a toda la aplicación.  **Vídeo** [**Tutorial**]**:** [*La clase Application en Android*](http://youtu.be/1Lr3civY9RI)*.*  Veamos cómo usarla en tres pasos:  1. Crea un descendiente de Applicaction que contenga la información global y los métodos asociados para acceder a esta información. Mira el ejemplo:  **public** **class** Aplicacion **extends** Application {  **private** **int** saldo;    @Override  **public** **void** onCreate() {  SharedPreferences pref = getSharedPreferences("pref", *MODE\_PRIVATE*);  **int** saldo = pref.getInt("saldo\_inicial", -1);  }    **public** **int** getSaldo(){  **return** saldo;  }    **public** **void** setSaldo(**int** saldo){  **this**.saldo=saldo;  }  }  En nuestra aplicación queremos que el usuario disponga de un saldo de puntos, con los que podrá ir desbloqueando ciertas características especiales. La clase Application es descendiente de Context, por lo que tendremos acceso a todos los métodos relativos a nuestro contexto. Entre estos métodos se incluye getSharedPreferences, para acceder a un fichero de preferencias almacenado en la memoria interna de nuestra aplicación. Además de poder sobrescribir el método onCreate(), la clase Application permite sobrescribir los siguientes:  onCreate() llamado cuando se cree la aplicación. Puedes usarlo para inicializar los datos.  onConfigurationChanged(Configuration nuevaConfig) llamado cuando se realicen cambios en la configuración del dispositivo, mientras que la aplicación se está ejecutando.  onLowMemory() llamado cuando el sistema se está quedando sin memoria. Trata de liberar toda la memoria que sea posible.  onTrimMemory(int nivel) (*desde nivel API 14*) llamado cuando el sistema determina que es un buen momento para que una aplicación recorte memoria. Esto ocurrirá, por ejemplo, cuando está en el fondo de la pila de actividades y no hay suficiente memoria para mantener tantos procesos en segundo plano. Además, se nos pasa como parámetro el nivel de necesidad. Algunos valores posibles son: TRIM\_MEMORY\_COMPLETE, TRIM\_MEMORY\_BACKGROUND, TRIM\_MEMORY\_MODERATE, …  2. Registra la clase creada en *AndroidManifest*. Para ello busca la etiqueta <application> y añade el atributo name, con el nombre de la clase creada:  ...  <application  **android:name=*"Aplicacion"***  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"*>  ...  3. Puedes obtener una referencia a tu clase Application con este código:  Aplicacion aplicacion = (Aplicacion) contexto.getApplication();   Donde contexto es una referencia a la clase Context. En caso de estar en un descendiente de esta clase (como Activity, Service,…) no es necesario disponer de esta referencia, la misma clase ya es un Context. Por lo tanto, podríamos escribir:  Aplicacion aplicacion = (Aplicacion) getApplication();   o incluso directamente:  **int** miSaldo = ((Aplicacion) getApplication()).getSaldo();  **Ejercicio paso a paso:** *Acceso a información global con la clase Application.*  **1.** Crea un nuevo proyecto con los siguientes datos:  Application Name: Audiolibros  Project Name: Audiolibrosv1  Package name: com.example.audiolibros  Minimum Required SDK: API 8 (2.2)  Utiliza los valores por defecto en el resto de los campos.  **2.** Vamos a empezar creando una nueva clase. Para ello pulsa con el botón derecho sobre el nombre del paquete y selecciona *New / Class*. Introduce como nombre de la clase Aplicacion. En ella vamos a almacenar dos objetos que queremos usar globalmente en toda la aplicación, vectorLibros y adaptador. Reemplaza su código por el siguiente:  **public** **class** Aplicacion **extends** Application {    **private** Vector<Libro> vectorLibros;  **private** AdaptadorLibros adaptador;    @Override  **public** **void** onCreate() {  vectorLibros = Libro.ejemploLibros();  adaptador = **new** AdaptadorLibros (**this**, vectorLibros);  }    **public** AdaptadorLibros getAdaptador() {  **return** adaptador;  }    **public** Vector<Libro> getVectorLibros() {  **return** vectorLibros;  }  }  ***Nota:*** *Tras incluir nuevas clases tendrás que indicar los imports adecuados. Para que Eclipse lo haga automáticamente pulsa «Ctrl+Shift+O».*  Aparecerán algunos errores dado que las clases Libro y AdaptadorLibros aún no han sido creadas. No te preocupes, lo haremos más adelante.  **3.** Registra el nombre de la clase en *AndroidManifest*, añadiendo la línea que aparece en negrita.  <application  **android:name=*"Aplicacion"***  android:allowBackup=*"true"*  ...  **4.** El siguiente paso va a ser crear la clase Libro, que definirá las características de cada audiolibro. Para ello usa el siguiente código:  public class Libro {  public String titulo;  public String autor;  public int recursoImagen;  public String urlAudio;  public String genero; // Género literario  public Boolean novedad; // Es una novedad  public Boolean leido; // Leído por el usuario  public final static String *G\_TODOS* = "Todos los géneros";  public final static String *G\_EPICO* = "Poema épico";  public final static String *G\_S\_XIX* = "Literatura siglo XIX";  public final static String *G\_SUSPENSE* = "Suspense";  public final static String[] *G\_ARRAY* = new String[] {*G\_TODOS*, *G\_EPICO*,  *G\_S\_XIX*, *G\_SUSPENSE* };    public Libro(String titulo, String autor, int recursoImagen,  String urlAudio, String genero, Boolean novedad, Boolean leido) {  this.titulo = titulo; this.autor = autor;  this.recursoImagen = recursoImagen; this.urlAudio = urlAudio;  this.genero = genero; this.novedad = novedad; this.leido = leido;  }    public static Vector<Libro> ejemploLibros() {  final String SERVIDOR = "http://www.dcomg.upv.es/~jtomas/android/audiolibros/";  Vector<Libro> libros = new Vector<Libro>();  libros.add(new Libro("Kappa", "Akutagawa",  R.drawable.*kappa*, SERVIDOR+"kappa.mp3",  Libro.*G\_S\_XIX*, false, false));  libros.add(new Libro("Avecilla", "Alas Clarín, Leopoldo",  R.drawable.*avecilla*, SERVIDOR+"avecilla.mp3",  Libro.*G\_S\_XIX*, true, false));  libros.add(new Libro("Divina Comedia", "Dante",  R.drawable.*divinacomedia*, SERVIDOR+"divina\_comedia.mp3",  Libro.*G\_EPICO*, true, false));  libros.add(new Libro("Viejo Pancho, El", "Alonso y Trelles, José",  R.drawable.*viejo\_pancho*, SERVIDOR+"viejo\_pancho.mp3",  Libro.*G\_S\_XIX*, true, true));  libros.add(new Libro("Canción de Rolando", "Anónimo",  R.drawable.*cancion\_rolando*, SERVIDOR+"cancion\_rolando.mp3",  Libro.*G\_EPICO*, false, true));  libros.add(new Libro("Matrimonio de sabuesos", "Agata Christie",  R.drawable.*matrimonio\_sabuesos*,SERVIDOR+"matrim\_sabuesos.mp3",  Libro.*G\_SUSPENSE*, false, true));  libros.add(new Libro("La iliada", "Homero",  R.drawable.*iliada*, SERVIDOR+"la\_iliada.mp3",  Libro.*G\_EPICO*, true, false));  return libros;  }  }  ***Nota:*** *Tras incluir nuevas clases tendrás que indicar los imports adecuados. Para que Eclipse lo haga automáticamente pulsa «Ctrl+Shift+O».*  Observa cómo se han añadido diferentes campos para definir las características de un audiolibro. La carátula del libro se almacena, en local, como un recurso de imagen, mientras que se accede a su audio por medio de una URL. El constructor nos permite inicializar todos los campos del audiolibro. También se ha incluido el método estático ejemploLibros() que nos devuelve un vector de audiolibros que contiene varios ejemplos.  **5.** Nos falta añadir en los recursos las carátulas de los libros. Comienza creando la carpeta *res/drawable.* Para ello pulsa con el botón derecho sobre la carpeta *res* del proyecto y selecciona*New / Folder*. Introduce como nombre de la carpeta drawable. Descarga las imágenes de la siguiente URL <http://www.dcomg.upv.es/~jtomas/android/audiolibros/imagenes> y cópialas en la carpeta creada.  **6.** La clase AdaptadorLibros no ha sido declarada. Lo haremos en el siguiente apartado. Por lo tanto, no podremos ejecutar la aplicación hasta terminar ese apartado. |

## 1.2.2. Uso del patrón Singleton

|  |
| --- |
| ***Nota:*** *Se trata de una sección avanzada. Te recomendamos te saltes este apartado en una primera lectura.*  **Vídeo** [**Tutorial**]**:** [*El patrón de diseño Singleton*](http://youtu.be/qxfhz1gk79g)*.*  Otra alternativa para almacenar información global a una aplicación es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia, a la que se podrá acceder desde cualquier clase utilizando un método estático. Una posible implementación de este patrón en Java se muestra a continuación:  **public** **class** Singleton {    // Esta será la instancia única de esta clase  **private** **static** Singleton *INSTANCIA* = **new** Singleton();    // El constructor es private para evitar su acceso desde fuera.  **private** Singleton() {}    // Método para obtener la única instancia de la clase  **public** **static** Singleton getInstancia() {  **return** *INSTANCIA*;  }  }  Para obtener la instancia de la clase escribimos desde cualquier sitio:  Singleton referencia = Singleton.getInstancia();  De hecho, esta es la única forma de acceder a la clase. Al no disponer de constructores públicos no podremos crear nuevos objetos.  Veamos cómo se implementaría el ejemplo anterior, donde se almacenaba el saldo de una aplicación, pero esta vez con el patrón Singleton. El primer problema que se nos plantea es la necesidad de disponer del contexto de la aplicación para poder acceder a las preferencias. Como el *Singleton* no dispone de constructor donde indicar esta información, nos vemos obligados a crear un método para su inicialización (inicializa()).  public class Saldo {  private static Saldo *INSTANCIA* = new Saldo();  // En Android casi siempre necesitas conocer el contexto  private Context contexto;  // Otras variables de la clase  private int saldo = -1;    private Saldo() {}    public static Saldo getInstancia() {  return *INSTANCIA*;  }    // Método para inicializar el objeto  public void inicializa(Context contexto){  this.contexto = contexto;  SharedPreferences pref = contexto.getSharedPreferences("pref",  Context.*MODE\_PRIVATE*);  int saldo = pref.getInt("saldo\_inicial", -1);  }    public int getSaldo() {  return saldo;  }    public void putSaldo(int saldo) {  this.saldo = saldo;  }  }  Para utilizar esta clase puedes usar el siguiente código:  Saldo saldo = Saldo.getInstancia();  saldo.inicializa(contexto);  int n = saldo.getSaldo();  Pero cuidado, asegúrate de llamar a inicializa() antes de usarla, y luego no volver a llamar a este método.  **Práctica:** *Acceso a información global con el patrón Singleton.*  **1.** Crea una nueva clase con nombre InfoGlobal que siga el patrón *Singleton*.  **2.** Añade como variables privadas los objetos vectorLibros y adaptador.  **3.** Añade el método inicializa(Context) y dos métodos *getter*.  **4.** Reemplaza, en toda la aplicación, los objetos vectorLibros y adaptador por un acceso adecuado a esta clase.  Aunque la clase Application fue creada para almacenar información global a la aplicación, desde la misma documentación de Android se nos recomienda utilizar el patrón *Singleton* para este propósito:  *“There is normally no need to subclass Application. In most situations, static singletons can provide the same functionality in a more modular way”*  Es decir, la clase Saldo, basada en patrón *Singleton*, tiene una mayor modularidad que la clase Aplicacion, descendiente de Application, dado que puede ser reutilizada en otros proyectos sin ser modificada. Aunque, el uso de Application también tendría sus ventajas: un código más limpio y que no requiere ser inicializado desde fuera de la clase. Dejamos en manos del lector la decisión de utilizar uno u otro mecanismo. |

# 1.3. GridView

|  |
| --- |
| Un GridView es un tipo de *layout* que nos muestra una serie de elementos en forma de tabla y que podemos deslizar verticalmente. El GridView representa una cuadrícula de vistas, cada una de ellas puede mostrar diversos elementos. Estas vistas podrán ser desplazadas arriba y abajo.  C:\Users\damon\Desktop\20141403021415_s4_image001.png  Para definir un GridView se requieren cuatro elementos. Para empezar, deberemos introducir el GridView en un *layout* de una actividad. En segundo lugar, necesitaremos diseñar un *layout* para los elementos que se muestran dentro del GridView. En tercer lugar, necesitaremos una actividad que lo muestre y lo controle. Esta actividad también será la que recibirá los eventos generados por el GridView. Por último, precisaremos un adaptador, que cree cada elemento del GridView y rellene la información necesaria de este. El adaptador utilizará el *layout* definido en el segundo paso y lo modificará con la información de cada elemento.  El uso de GridView es muy parecido al de ListView. Puedes encontrar información sobre el uso de ListView en [androidcurso.com](http://androidcurso.com/index.php/128)[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student" \l "_ftn1" \o ") o en el tercer capítulo de*El gran libro de Android*.  **Ejercicio paso a paso:** Primera versión de Audiolibros con un GridView.  En este ejercicio completaremos la aplicación Audiolibros con un GridView que nos mostrará la lista de libros, permitiendo al usuario seleccionar uno de ellos.  C:\Users\damon\Desktop\20141403021415_s4_image003.jpg  Para empezar, necesitaremos definir un GridView en el *layout* a mostrar. Modifica *res/layout/activity\_main.xml* para que tenga el siguiente código:  <GridView xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:id=*"@+id/gridview"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  android:columnWidth=*"120dp"*  android:numColumns=*"auto\_fit"*  android:verticalSpacing=*"10dp"*  android:horizontalSpacing=*"10dp"*  android:gravity=*"top"*  android:background=*"#FFFFFF"*  />  Como puedes ver, el *layout* contiene solo un elemento: el GridView. Veamos qué significan los atributos indicados. Con el primero, columnWidth, estamos indicando al GridView el ancho de las columnas; esta propiedad va emparejada con la siguiente, numColumns, que indica la cantidad de columnas a mostrar. Para estas propiedades tenemos dos posibilidades: dar un ancho fijo para cada columna y dejar que el sistema ajuste la cantidad de columnas, o indicar la cantidad de columnas y dejar al sistema el ancho de estas. Las siguientes dos propiedades indican el margen a dejar entre elementos.  Ahora modifica la clase de la actividad principal, MainActivity, para que contenga el siguiente código:  **public** **class** MainActivity **extends** Activity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  GridView gridview = (GridView) findViewById(R.id.*gridview*);  Aplicacion app = (Aplicacion) activity.getApplication();  gridview.setAdapter(app.getAdaptador());  gridview.setOnItemClickListener(**new** OnItemClickListener() {  **public** **void** onItemClick(AdapterView<?> parent, View v, **int** position, **long** id) {  Toast.*makeText*(MainActivity.**this**, "Seleccionado el elemento: " + position, Toast.*LENGTH\_SHORT*).show();  }  });  }  }  ***Nota:*** *Tras incluir nuevas clases tendrás que indicar los imports adecuados. Para que Eclipse lo haga automáticamente pulsa «Ctrl+Shift+O».*  En esta clase, lo único que hacemos es asignar el *layout* a la actividad, buscar en él el GridView y asignarle el adaptador que se definió en la clase Aplicacion. Finalmente, crearemos un escuchador de eventos, para que cuando se pulse sobre un elemento, muestre en una notificación *toast* el elemento que se ha pulsado.  El siguiente paso será crear la clase AdaptadorLibros, que se encargará de rellenar el GridView. Crea esta clase con el siguiente código:  **public** **class** AdaptadorLibros **extends** BaseAdapter {  **private** LayoutInflater inflador; // Crea Layouts a partir del XML  **protected** Vector<Libro> vectorLibros; // Vector con libros a visualizar    **public** AdaptadorLibros(Context contexto, Vector<Libro> vectorLibros) {  inflador = (LayoutInflater) contexto.getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);  **this**.vectorLibros = vectorLibros;  }  **public** View getView(**int** posicion, View vistaReciclada, ViewGroup padre) {  Libro libro = vectorLibros.elementAt(posicion);  **if** (vistaReciclada == **null**) {  vistaReciclada = inflador.inflate(R.layout.*elemento\_selector*, **null**);  }  TextView titulo = (TextView) vistaReciclada.findViewById(R.id.*titulo*);  titulo.setText(libro.titulo);  ImageView portada = (ImageView) vistaReciclada.findViewById(R.id.*portada*);  portada.setImageResource(libro.recursoImagen);  portada.setScaleType(ImageView.ScaleType.*CENTER\_INSIDE*);  **return** vistaReciclada;  }    **public** **int** getCount() {  **return** vectorLibros.size();  }    **public** Object getItem(**int** posicion) {  **return** vectorLibros.elementAt(posicion);  }    **public** **long** getItemId(**int** posicion) {  **return** posicion;  }  }  La clase BaseAdapter es un mecanismo estándar en Android que nos permite crear una serie de vistas que han de ser mostradas dentro de un contenedor. Además de emplearse en GridView, también se utiliza para crear un ListView o un Spiner.  En esta clase el método más importante es getView(), que usa el sistema para pedir cada uno de los elementos a insertar. Cuando llamen a getView(), nos indicarán tres parámetros: la posición del elemento a insertar, una vista reciclada y el *layout* contenedor donde se insertará el elemento. Este método ha de devolver una vista con la información adecuada del elemento a insertar.  El parámetro vistaReciclada se utiliza para mejorar el rendimiento en la creación de vistas. Para la primera llamada a getView(), este parámetro será nulo y tendremos que crear una nueva vista e inflarla desde un XML (este proceso puede ser algo lento). Pero para las siguientes llamadas, este parámetro contendrá la vista devuelta por nosotros en la llamada anterior, para esta posición. De esta forma ya no será necerario crearla desde cero y solo tendremos que modificar sus características y devolverla.  ***Nota:*** *En este ejemplo el* BaseAdapter *devuelve siempre el mismo tipo de vista. Aunque esta es la forma más habitual de trabajar, no tiene por qué ser necesariamente así. Cada elemento devuelto desde* getView() *podría ser una vista totalmente diferente a las demás.*  Finalmente tenemos que definir tres métodos que permiten acceder a la información representada. El método getCount() será el que indique cuántos elementos queremos mostrar. El método getItem() devolverá el objeto libro que se muestra en una determinada posición. El método getItemId() devolverá el *id* libro que se muestra en una determinada posición. El *id* es un valor numérico que identifica cada elemento. Si utilizamos un vector se suele utilizar el índice del vector como *id*, mientras que en bases de datos suele ser el campo de indexación principal. En esta primera versión del adaptador se muestran todos los libros en el mismo orden en que los tenemos en el vector, por lo tanto, el *id* coincide con la posición. Más adelante mostraremos un ejemplo más complejo donde ya no ocurrirá esto.  Por último, solo tenemos que crear el *layout* que utiliza nuestro adaptador como base para crear las diferentes vistas del GridView. Pulsa con el botón derecho en el explorador del proyecto y selecciona*New/Android XML File*. El nombre del fichero ha de ser *elemento\_selector.xml*. Reemplaza su código por el siguiente:  <RelativeLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  android:orientation=*"vertical"* >  <ImageView  android:id=*"@+id/portada"*  android:layout\_width=*"120dp"*  android:layout\_height=*"180dp"*  android:layout\_alignParentTop=*"true"*  android:layout\_centerHorizontal=*"true"*  android:layout\_margin=*"12dp"*  android:src=*"@drawable/books"* />  <ImageView  android:id=*"@+id/sombra"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"wrap\_content"*  android:src=*"@drawable/sombra0"*  android:layout\_alignBottom=*"@+id/portada"*  android:layout\_alignParentLeft=*"true"*  android:layout\_marginBottom=*"-21dp"* />  <TextView  android:id=*"@+id/titulo"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"wrap\_content"*  android:text=*"Título"*  android:layout\_alignParentLeft=*"true"*  android:layout\_below=*"@id/portada"*  android:gravity=*"center"*  android:lines=*"2"*  android:textColor=*"#000000"*  android:textSize=*"18dp"* />  </RelativeLayout>  Ejecuta la aplicación para ver el resultado.  **Preguntas de repaso:**[*GridView*](http://www.dcomg.upv.es/~jtomas/android/test_avanzado/test1_1GridView.html)*.*  [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student" \l "_ftnref1" \o ") <http://androidcurso.com/index.php/128> |

# 1.4. Fragments

|  |
| --- |
| **Vídeo** [**Tutorial**]**:** [*Los Fragments en Android*](http://youtu.be/MZvms25uDU8)*.*  Con la versión 3.0 de Android, diseñada específicamente para tabletas, surgió un problema: las diferencias en los tamaños de pantalla entre los móviles y las tabletas complicaban los diseños de la interfaz de usuario. En una tableta, pueden caber muchos elementos de diseño al mismo tiempo, mientras que en un móvil estamos más limitados. Para resolver este problema, en la versión 3.0 de Android, se introdujeron los *fragments*. Los *fragments* son bloques de interfaz de usuario que pueden utilizarse en diferentes sitios, simplificando así la composición de una interfaz de usuario.  Los *fragments* nos permiten diseñar y crear cada uno de los elementos de nuestra aplicación por separado. Luego, dependiendo del tamaño de pantalla disponible, mostraremos uno solo, o más de uno a la vez. Por ejemplo, podríamos diseñar dos *fragments*, uno que nos permitiera elegir entre una lista de libros y otro que mostrara los detalles de uno de estos libros. En una tableta se podría mostrar ambos *fragments* a la vez mientras que en un móvil tendríamos que mostrar uno, y luego el otro.  C:\Users\damon\Desktop\20141403021416_s5_image002.jpg  ***Figura 1.*** *Uso de* fragments *en tableta y móvil.*  Es importante resaltar que no cambia el papel de las actividades. Sigue siendo el elemento básico que representa cada pantalla de una aplicación y nos permite navegar por ella. La novedad introducida es que cuando diseñemos una actividad, esta puede estar formada por uno o más *fragments*.  Cuando diseñemos un *fragment*, este ha de gestionarse a sí mismo, recibiendo eventos de entrada y modificando su vista sin necesidad de que la actividad que lo contiene intervenga. De esta forma, el *fragment* se podrá utilizar en diferentes actividades sin tener que modificar el código.  Los *fragments* son muy importantes dado que a partir de ahora Android los utiliza como elemento base en el diseño de la interfaz de usuario. Por ejemplo, la última versión de Google Maps o la visualización de preferencias de usuario se basan en *fragments*. El problema es que esta característica aparece en una versión que todavía no está disponible en muchos dispositivos. Para resolver este problema se ha creado una librería de compatibilidad para poder utilizar*fragments* en versiones anteriores a la 3.0. Esta librería se incluye de manera automática a un proyecto, siempre que el requerimiento mínimo de SDK sea inferior al nivel 11 (3.0); pero lo desarrollemos con una versión superior a la 3.0 (*Target SDK*). Para verificar esto, abre el proyecto creado en el ejercicio anterior. Observa cómo esta librería se incluye en *libs/android-support-v4.jar*.  Cada *fragment* ha de implementarse en una clase diferente. Esta clase tiene una estructura similar a la de una actividad, pero con algunas diferencias. La primera es que esta clase tiene que extender Fragment. El ciclo de vida es muy parecido al de una actividad; sin embargo, dispone de unos cuantos eventos más, que le indican cambios en su estado con respecto a la actividad que lo contiene. El ciclo de vida de un *fragment* va asociado al de la actividad que lo contiene (por ejemplo, si la actividad es destruida, todos los *fragments* que contiene son destruidos); pero también es posible destruir un *fragment* sin modificar el estado de la actividad.  Los *fragments* suelen mostrar una vista (aunque esto no es imprescindible). Es recomendable definir esta vista en un fichero XML de recursos. Por lo tanto, para crear un *fragment* usaremos una clase Java para definir su comportamiento y un fichero XML para definir su apariencia.  Los *fragments* se pueden introducir en una actividad de dos formas diferentes: por código o desde XML. Ambas formas tienen sus ventajas y sus inconvenientes. Introducir un *fragment* desde XML es más sencillo. Además, el diseño queda diferenciado del código, simplificando el trabajo del diseñador. Sin embargo, trabajar de esta forma tiene un inconveniente: una vez introducido ya no podremos reemplazar el *fragment* por otro. Por lo tanto, un *fragment* añadido desde XML será siempre estático. Si lo añadimos desde código, ganamos la posibilidad de intercambiar el *fragment* por otro. En los siguientes ejercicios veremos cómo añadir *fragments* de las dos formas. |

## 1.4.1. Insertar fragments desde xml

|  |
| --- |
|  |

|  |
| --- |
| **Ejercicio paso a paso:** Un primer ejemplo con Fragments.  En este ejercicio modificaremos el ejercicio anterior que visualizaba una lista de libros, pero ahora trabajando con *fragments*. Su funcionalidad será idéntica. El objetivo es ver cómo se define un *fragment* sencillo y cómo se utiliza una vez creado.  **1.** Crea una copia del ejercicio anterior, cambiando el nombre de proyecto desde *Audiolibrosv1* a *Audiolibrosv2*. Para ello, haz clic con el botón derecho en la carpeta de proyecto que aparece en la vista *Package Explorer* y selecciona *Copy*. Entonces, haz clic con el botón derecho en el mismo sitio y selecciona *Paste*. Te pedirá el nombre de proyecto; indica *Audiolibrosv2*.  **2.** Como el único *fragment* que vamos a mostrar contendrá lo que antes mostraba MainActivity, podemos reutilizar su *layout* en XML para nuestro *fragment.* Renombra el fichero*activity\_main.xml* por *fragment\_selector.xml*. Para ello, haz clic con el botón derecho en él y selecciona *Refactor/Rename*.  Ahora nos falta definir la clase para el *fragment*. Para que el código quede organizado, colocaremos los *fragments* en otro paquete. Esto es totalmente opcional; pero en nuestro caso nos ayudará a mantener por separado los *fragments* de otras clases. Para crear otro paquete pulsa con el botón derecho sobre la carpeta *src* y selecciona *new/package* e indica el nombre*com.example.audiolibros.fragments*.  Crea una nueva clase dentro de este paquete llamada SelectorFragment y rellénala con el siguiente código:  **public** **class** SelectorFragment **extends** Fragment {  **private** Activity actividad;  **private** GridView gridview;  **private** AdaptadorLibros adaptador;  **private** Vector<Libro> vectorLibros;    @Override  public void onAttach(Activity actividad) {  **super**.onAttach(actividad);  **this**.actividad = actividad;  Aplicacion app = (Aplicacion) actividad.getApplication();  adaptador = app.getAdaptador();  vectorLibros = app.getVectorLibros();  }    @Override  **public** View onCreateView(LayoutInflater inflador, ViewGroup contenedor,  Bundle savedInstanceState) {  View vista = inflador.inflate(R.layout.*fragment\_selector*, contenedor, **false**);  gridview = (GridView) vista.findViewById(R.id.*gridview*);  gridview.setAdapter(adaptador);  gridview.setOnItemClickListener(**new** OnItemClickListener() {  **public** **void** onItemClick(AdapterView<?> parent, View v, **int** posicion, **long** id) {  Toast.*makeText*(actividad,"Seleccionado el elemento: "+posicion,Toast.*LENGTH\_SHORT*).show();  }  });  **return** vista;  }  }  ***Nota:*** *Tras incluir nuevas clases tendrás que indicar lo imports adecuados. Para que Eclipse lo haga automáticamente pulsa «Ctrl+Shift+O». La clase R aparece en dos paquetes, por lo que te pedirá que selecciones uno de los dos:*  C:\Users\damon\Desktop\20141403021416_s6_image002.jpg    En este caso selecciona el paquete de tu aplicación, para así poder acceder a tus recursos al utilizar el identificador R.  Lo mismo ocurre con la clase fragment que se encuentra en dos paquetes.  C:\Users\damon\Desktop\20141403021416_s6_image003.jpg  Utiliza el segundo que corresponde a la librería de compatibilidad. El primero has de gastarlo solo cuando trabajes con una versión mínima igual o superior a la 3.0 y por lo tanto no necesites la librería de compatibilidad. Es muy importante que utilices el mismo paquete en todo el proyecto. Aunque posiblemente la definición de un fragment en los dos paquetes sea idéntica, para Java se trata de dos clases distintas.  El código de esta clase es similar al que teníamos antes en MainActivity, salvo que ahora extendemos a Fragment en vez de a Activity, y que los métodos del ciclo de vida son diferentes. Estudiaremos más adelante el ciclo de vida de un fragment. De momento comentar que el método onAttach() es llamado cuando el fragment se asocia a la actividad. En él se nos pasa una referencia a la actividad, que guardaremos para usarla más adelante. También aprovecharemos este método para obtener dos referencias a los dos objetos declarados en Aplicacion. De igual manera como en una actividad un fragment también tiene una vista asociada. En la actividad asociábamos la vista en el método onCreate(), llamando a setContentView(). En un fragmenttambién disponemos del método onCreate(), pero no es aquí donde hay que asociar la vista. Se ha creado un nuevo método en el ciclo de vida, onCreateView(), con la finalidad de asociar su vista. En este método se nos pasan tres parámetros: un LayoutInflater que nos permite crear una vista a partir de un layout XML, el contenedor donde será insertado el fragment (en el punto siguiente veremos que se trata de un LinearLayout) y posibles valores guardados de una instancia anterior[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftn1). El método onCreateView() ha de devolver la vista ya creada. El hecho de disponer de estos dos métodos va a resultar muy interesante, dado que nos va a permitir cambiar la vista de un fragment sin tener que volverlo a crear.  Crea de nuevo un fichero XML para MainActivity llamado activity\_main.xml. El fichero tendrá que contener el siguiente código:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  android:orientation=*"horizontal"* >  <fragment  android:id=*"@+id/selector\_fragment"*  android:name=*"com.example.audiolibros.fragments.SelectorFragment"*  android:layout\_width=*"0dp"*  android:layout\_height=*"match\_parent"*  android:layout\_weight=*"1"* />  </LinearLayout>  Como puedes ver, introducir el *fragment* desde un XML es muy sencillo. Simplemente añadimos una etiqueta <fragment> y en el atributo name le indicamos el nombre de la clase del *fragment.*  Modifica la clase MainActivity para que contenga el siguiente código:  **public class** MainActivity **extends** FragmentActivity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  }  }  La clase Activity antes de la versión 3.0 no está preparada para contener *fragments*. Por esta razón utilizaremos la clase FragmentActivity, que se incluye en la librería de compatibilidad. Cuando trabajes con una versión mínima igual o superior a la 3.0 utiliza directamente la clase Activity.  Ejecuta el proyecto. Verifica que la aplicación tiene la misma funcionalidad que antes.  **Ejercicio paso a paso:** *Implementando un segundo fragment.*  Recordemos que la aplicación que queremos hacer tiene que mostrar una serie de audiolibros para elegir, y que al pulsar sobre uno de ellos, nos muestre la información detallada sobre él y nos permita escucharlo. Para esto, vamos a crear un segundo *fragment* para ver los detalles de un audiolibro.   1. La vista que mostraremos estará definida en *fragment\_detalle.xml*. Crea este archivo dentro de *res/layout* y complétalo con el siguiente código:   <?xml version=*"1.0"* encoding=*"utf-8"*?>  <RelativeLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  xmlns:tools=*"http://schemas.android.com/tools"*  android:id=*"@+id/main\_fragment\_detalle"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  tools:context=*".MainActivity"* >  <TextView  android:id=*"@+id/titulo"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:layout\_alignParentLeft=*"true"*  android:layout\_alignParentRight=*"true"*  android:layout\_alignParentTop=*"true"*  android:gravity=*"center"*  android:text=*"Título"*  android:textAlignment=*"center"*  android:textAppearance=*"?android:attr/textAppearanceLarge"* />  <TextView  android:id=*"@+id/autor"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:layout\_alignParentLeft=*"true"*  android:layout\_alignParentRight=*"true"*  android:layout\_below=*"@+id/titulo"*  android:gravity=*"center"*  android:text=*"Autor"*  android:textAlignment=*"center"*  android:textAppearance=*"?android:attr/textAppearanceMedium"* />  <ImageView  android:id=*"@+id/portada"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:layout\_below=*"@id/autor"*  android:layout\_centerHorizontal=*"true"*  android:src=*"@drawable/ic\_launcher"* />  </RelativeLayout>  **2.** Modifica *activity\_main.xml* para que muestre ambos *fragments*. Para ello, añade el siguiente elemento al linear *layout*:  <fragment  android:id=*"@+id/detalle\_fragment"*  android:name=*"com.example.audiolibros.fragments.DetalleFragment"*  android:layout\_width=*"0dp"*  android:layout\_height=*"match\_parent"*  android:layout\_weight=*"1"* />  **3.** Crea una nueva clase llamada DetalleFragment dentro del paquete creado para los *fragments*. Esta clase deberá contener el siguiente código:  **public** **class** DetalleFragment **extends** Fragment **implements** OnTouchListener, OnPreparedListener, MediaController.MediaPlayerControl {  **public** **static** String *ARG\_ID\_LIBRO* = "id\_libro";  MediaPlayer mediaPlayer;  MediaController mediaController;    @Override  **public** View onCreateView(LayoutInflater inflador, ViewGroup contenedor,  Bundle savedInstanceState) {  View vista = inflador.inflate(R.layout.*fragment\_detalle*, contenedor, **false**);  Bundle args = getArguments();  **if** (args != **null**) {  **int** position = args.getInt(*ARG\_ID\_LIBRO*);  ponInfoLibro(position, vista);  } **else** {  ponInfoLibro(0, vista);  }  **return** vista;  }    **private** **void** ponInfoLibro(**int** id, View vista) {  Libro libro = ((Aplicacion) getActivity().getApplication())  .getVectorLibros().elementAt(id);  ((TextView) vista.findViewById(R.id.*titulo*)).setText(libro.titulo);  ((TextView) vista.findViewById(R.id.*autor*)).setText(libro.autor);  ((ImageView) vista.findViewById(R.id.*portada*)).setImageResource(libro.recursoImagen);  vista.setOnTouchListener(**this**);  **if** (mediaPlayer != **null**){  mediaPlayer.release();  }  mediaPlayer = **new** MediaPlayer();  mediaPlayer.setOnPreparedListener(**this**);  mediaController = **new** MediaController(getActivity());  Uri audio = Uri.*parse*(libro.urlAudio);  **try** {  mediaPlayer.setDataSource(getActivity(), audio);  mediaPlayer.prepareAsync();  } **catch** (IOException e) {  Log.*e*("Audiolibros", "ERROR: No se puede reproducir "+audio,e);  }  }    **public** **void** ponInfoLibro(**int** id) {  ponInfoLibro(id, getView());  }    @Override  **public** **void** onPrepared(MediaPlayer mediaPlayer) {  Log.*d*("Audiolibros", "Entramos en onPrepared de MediaPlayer");  mediaPlayer.start();  mediaController.setMediaPlayer(**this**);  mediaController.setAnchorView(getView().findViewById(R.id.*main\_fragment\_detalle*));  mediaController.setEnabled(**true**);  mediaController.show();  }    @Override  **public** **boolean** onTouch(View vista, MotionEvent evento) {  mediaController.show();  **return** **false**;  }    @Override  **public** **void** onStop() {  **super**.onStop();  **try** {  mediaPlayer.stop();  mediaPlayer.release();  } **catch** (Exception e) {  Log.*d*("Audiolibros", "Error en mediaPlayer.stop()");  }  }    @Override  **public** **boolean** canPause() {  **return** **true**;  }    @Override  **public** **boolean** canSeekBackward() {  **return** **true**;  }    @Override  **public** **boolean** canSeekForward() {  **return** **true**;  }    @Override  **public** **int** getBufferPercentage() {  **return** 0;  }    @Override  **public** **int** getCurrentPosition() {  **return** mediaPlayer.getCurrentPosition();  }    @Override  **public** **int** getDuration() {  **return** mediaPlayer.getDuration();  }    @Override  **public** **boolean** isPlaying() {  **return** mediaPlayer.isPlaying();  }    @Override  **public** **void** pause() {  mediaPlayer.pause();  }    @Override  **public** **void** seekTo(**int** pos) {  mediaPlayer.seekTo(pos);  }    @Override  **public** **void** start() {  mediaPlayer.start();  }    @Override  **public** **int** getAudioSessionId() {  **return** 0;  }  }  La finalidad de este *fragment* es visualizar los detalles de un audiolibro y permitir su reproducción. Para la reproducción declaramos el objeto mediaPlayer y para que el usuario pueda poner pausa, pararlo o mover el punto de reproducción declaramos el objeto mediaControler.  En el método onCreateView() vamos a generar la vista a representar. Comenzamos inflando el XML definido en el primer punto de este ejercicio. Luego, miramos si se han pasado argumentos al *fragment* mediante el método getArguments(). Si nos han pasado el argumento “id\_libro”, visualizaremos el audiolibro con este *id*, si no, visualizaremos el que tiene *id* 0.  El método ponInfoLibro() es el que se encarga de introducir la información del libro en la vista. Primero obtenemos una referencia del libro a representar, haciendo uso de la clase Aplicaciondefinida al principio del capítulo. El siguiente paso consiste en actualizar la información del título, autor y portada dentro de la vista. Luego, activamos un escuchador de evento onTouch, para cuando se pulse sobre esta vista. Finalmente, son configurados los objetos mediaPlayer y mediaControler. El método ponInfoLibro() dispone de una sobrecarga donde solo se indica el *id* del libro a visualizar, de forma que se modifique la vista actual del *fragment*.  Dado que el audio es reproducido desde Internet, hay que prepararlo de forma asíncrona (prepareAsync()). Si utilizáramos el método síncrono (prepare()), podríamos dejar la aplicación bloqueada demasiado tiempo[[2]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student" \l "_ftn2" \o "). Actuando de esta manera, continuará la ejecución del programa y cuando se tenga preparado el audio para reproducir se ejecutará el método onPrepared(). Desde este método ponemos en reproducción el mediaPlayer y configuramos el mediaControler asociado a este. Aunque activamos la visualización del mediaController, este desaparecerá a los tres segundos.  El método onTouch() es ejecutado cuando se pulse sobre la vista. Queremos que se muestre el mediaControler. Permanecerá visible tres segundos y luego volverá a ocultarse. El resto de métodos corresponden a la interfaz MediaController.MediaPlayerControl donde se realizan las diferentes acciones de este control para asociarlo con el objeto mediaPlayer.  **4.** Dado que los audios se reproducen directamente desde Internet, es imprescindible solicitar el permiso correspondiente. Añade la siguiente línea a *AndroidManifest.xml*:  <uses-permission android:name=*"android.permission.INTERNET"*/>  **5.** Ejecuta la aplicación. Podrás ver como se muestran los dos *fragments* uno al lado del otro. Pasados unos segundos se ha de poner en reproducción el primer audiolibro.  [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftnref1) <http://www.youtube.com/watch?v=NMAJfqDOpBQ>  [[2]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student" \l "_ftnref2" \o ") <http://androidcurso.com/index.php/271> o <http://youtu.be/ekn9j2J9sos> |
|  |

## 1.4.2. Insertar fragments desde código

|  |  |
| --- | --- |
| |  | | --- | |  | |
|  |

|  |
| --- |
|  |
| Si ejecutas el proyecto del ejercicio anterior en diferentes dispositivos, comprobarás que tanto en una tableta como en un móvil la apariencia es similar.  C:\Users\damon\Desktop\20141403021416_s7_image001.pngC:\Users\damon\Desktop\20141403021416_s7_image002.png  Sobre una tableta mostrar los *fragmments* simultáneamente parece adecuado. Sin embargo, en un móvil, puede ocasionar que su tamaño sea excesivamente pequeño. En este caso, parece más adecuado mostrar primero el *fragment* de selección y una vez que el usuario ha seleccionado un libro mostrar su detalle.  Para conseguir este comportamiento, cuando estemos en una pantalla grande incluiremos dos *fragments* desde XML; pero si estamos en una pequeña, los intercambiaremos en tiempo de ejecución. Para diferenciar entre pantallas grandes y pequeñas contamos con los recursos alternativos (más información en el siguiente vídeo). En los siguientes dos ejercicios veremos cómo conseguirlo.  **Vídeo** [**Tutorial**]**:** [*Uso de recursos alternativos en Android*](http://youtu.be/jwzjK7M2WkQ)*.*  **Ejercicio paso a paso:** *Implementando un segundo fragment.*  **1.** Primero, crearemos la carpeta de recursos alternativos para las vistas en pantalla grande. Para esto, crea una nueva carpeta dentro de *res* llamada *layout-large*.  ***Nota:*** *Si no recuerdas los nombres de las carpetas para recursos alternativos, puedes usar la opción desde el botón derecho, New/Android XML File y seleccionar el tipo de recurso alternativo desde la segunda ventana.*  **2.** El *layout* *activity\_main.xml* que tenemos actualmente dentro de *res/layout* es el que querremos para pantallas grandes. Cópialo dentro de *res/layout-large*.  **3.** Introduce en *res/layout/activity\_main.xml* el siguiente código:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:id=*"@+id/contenedor\_pequeno"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  android:orientation=*"horizontal"* />  Este *layout* simplemente tiene un solo LinearLayout. En el se insertará un DetalleFragment o un SelectorFragment según nos interese*.* Para insertar un único elemento también podríamos usar otro tipo de *layout*, como FrameLayout o RelativeLayout.  **4.** Finalmente, vamos a añadir el *fragment* dentro de este *layout* desde código. Para ello, copia el siguiente código dentro del onCreate() de MainActivity, después de la llamada asetContentView():  **if** (findViewById(R.id.*contenedor\_pequeno*) != **null**) {  SelectorFragment primerFragment = **new** SelectorFragment();  getSupportFragmentManager().beginTransaction()  .add(R.id.*contenedor\_pequeno*, primerFragment).commit();  }  Cuando el programa se ejecute en un dispositivo grande, se cargará el *layout* que contiene los dos *fragments* directamente en XML. Además, el *id* contenedor\_pequeno no será creado, por lo que el código anterior no será ejecutado. Si se ejecuta en un dispositivo pequeño, se cargará el layout añadido en el punto 3 y se ejecutará el código añadido en el punto 4.  En este código creamos una instancia del *fragment* que vamos a añadir, comenzamos una transacción, término que se utiliza en *fragments* para referirse a una adición, intercambio o eliminación de *fragments*. En esta transacción añadimos el *fragment* creado, usando el método add(). Finalmente, llevamos a cabo la transacción, usando el método commit().  **5.** Ejecuta el proyecto en una tableta; se visualizarán dos *fragments*. Y en un móvil se visualizará solo uno. |

## 1.4.3. Comunicación e intercambio de fragments

|  |
| --- |
| En este apartado veremos cómo se pueden comunicar los *fragments* con la actividad que los contiene. También veremos qué hacer para poder intercambiar dos*fragments* entre sí desde la actividad.  En particular veremos cómo desde SelectorFragment podemos indicar a MainActivity que ha recibido una pulsación, cómo MainActivity puede intercambiarSelectorFragment por DetalleFragment y, finalmente, cómo MainActivity puede indicar a DetalleFragment el libro a mostrar.    **Ejercicio paso a paso:** Comunicación e intercambio de *fragments*.  **1.** En el método onCreateView() de selectorFragment sustituye las líneas:  gridview.setOnItemClickListener(  …  });  Por el siguiente código:  gridview.setOnItemClickListener(**new** OnItemClickListener(){  **public** **void** onItemClick(AdapterView<?> parent, View v, **int** posicion, **long** id) {  ((MainActivity) actividad).mostrarDetalle((**int**) id);  }  });  Cuando se pulse sobre un elemento, en lugar de mostrar un Toast, vamos a llamar a un método de nuestra actividad para que se nos muestre el detalle del libro seleccionado.  **2.** Añade el la clase MainActivity el siguiente método:  **public** **void** mostrarDetalle(**int** id) {  DetalleFragment detalleFragment = (DetalleFragment)  getSupportFragmentManager().findFragmentById(R.id.*detalle\_fragment*);    **if** (detalleFragment != **null**) {  detalleFragment.ponInfoLibro(id);  } **else** {  DetalleFragment nuevoFragment = **new** DetalleFragment();  Bundle args = **new** Bundle();  args.putInt(DetalleFragment.*ARG\_ID\_LIBRO*, id);  nuevoFragment.setArguments(args);  FragmentTransaction transaccion = getSupportFragmentManager().beginTransaction();  transaccion.replace(R.id.*contenedor\_pequeno*, nuevoFragment);  transaccion.addToBackStack(**null**);  transaccion.commit();  }  }  Comenzamos averiguando si existe el *fragment* detalleFragment. Si existe significa que estaremos en una tableta y solo tendremos que actualizar la información del libro a mostrar. Usaremos para ello el método ponInfoLibro(). En caso de existir, significa que estamos en un móvil. En este caso, crearemos un nuevo DetalleFragment, le pasaremos como argumento el *id* del libro a mostrar y crearemos una nueva transacción, esta vez con replace(), para reemplazar el *fragment* actual.  Un detalle que hay que destacar es cómo, antes del método commit(), llamamos a addToBackStack(). Lo que hace este método es guardar esta transacción en la pila de navegación entre actividades; así, cuando el usuario pulse la tecla «atrás», la transacción se deshará, en vez de cerrar la actividad. Desde el punto de vista del usuario, esto será como volver un paso atrás.  **3.** Ejecuta la aplicación en un dispositivo pequeño. Podrás ver que solo se muestra el *fragment* que contiene el GridView, y que al seleccionar un elemento, se sustituye este *fragment* por el que muestra los detalles del libro seleccionado. Si pulsas la tecla «atrás» mientras ves los detalles, se vuelve al *fragment* del GridView.  **4.** Comenta la línea transaction.addToBackStack(null) y ejecuta de nuevo la aplicación en un dispositivo pequeño. Cuando estés visualizando el detalle de un libro pulsa la tecla «atrás». Comprueba como da por finalizada la actividad y regresa a la aplicación anterior. Dado que no es el comportamiento esperado por el usuario, quita el comentario de la línea.      **Ejercicio paso a paso:***Añadiendo un menú contextual al GridView.*  En este ejercicio vamos a añadir un menú contextual que permita al usuario realizar varias acciones sobre su biblioteca de libros. En concreto permitiremos compartir, borrar o insertar libros.  C:\Users\damon\Desktop\20141403021416_s8_image003.jpg  **1.** Añade dentro de onCreateView() de SelectorFragment el siguiente código:  gridview.setOnItemLongClickListener(**new** OnItemLongClickListener() {  **public** **boolean** onItemLongClick(AdapterView<?> padre, **final** View v,**final** **int** posicion, **final** **long** id) {    AlertDialog.Builder menu = **new** AlertDialog.Builder(actividad);  CharSequence[] opciones = { "Compartir", "Borrar ", "Insertar" };  menu.setItems(opciones, **new** DialogInterface.OnClickListener() {  **public** **void** onClick(DialogInterface dialog, **int** opcion) {  **switch** (opcion) {  **case** 0: //Compartir  Libro libro = vectorLibros.elementAt((**int**) id);  Intent i = **new** Intent(Intent.*ACTION\_SEND*);  i.setType("text/plain");  i.putExtra(Intent.*EXTRA\_SUBJECT*, libro.titulo);  i.putExtra(Intent.*EXTRA\_TEXT*, libro.urlAudio);  startActivity(Intent.*createChooser*(i, "Compartir"));  **break**;  **case** 1: //Borrar  vectorLibros.remove((**int**) id);  adaptador.notifyDataSetChanged();  **break**;  **case** 2: //Insertar  vectorLibros.add(vectorLibros.elementAt((**int**) id));  adaptador.notifyDataSetChanged();  **break**;  }  }  });  menu.create().show();  **return** **true**;  }  });  Aquí estamos definiendo un OnItemLongClickListener() que utilizaremos para detectar una pulsación larga. Cuando ocurra este evento, mostraremos un menú contextual con las opciones de compartir, borrar o insertar. A su vez definimos un OnClickListener en el menú, para detectar las diferentes opciones.  En la primera opción permitimos compartir la URL del libro seleccionado por medio de la intención ACTION\_SEND. Se abrirá un menú para que el usuario seleccione alguno de los programas que tenga instalados que admitan esta intención; por ejemplo Facebook, Twitter o WhatsApp. La segunda opción elimina el libro seleccionado y notifica al adaptador que ha habido cambios para que redibuje el Gridview. La última opción crea una copia del libro seleccionado al final de la lista.  Terminamos mostrando el menú y devolviendo **true** para indicar que hemos procesado la pulsación larga y no hay que seguir propagando el evento.  ***Nota:*** *Observa como los parámetros del método onItemLongClick() tienen el modificador* ***final****. Si lo eliminas, por ejemplo del parámetro id, aparece el error: “Cannot refer to a non-final variable id inside an inner class defined in a different method”. Los parámetros de un método se comportan como variables locales. Es decir, son eliminados cuando el método termina. Sin embargo, este método crea un objeto OnClickListener que permanecerá activo una vez el método haya terminado. Y desde este objeto accedemos a un parámetro del método. Para asegurarnos que este parámetro no es destruido al terminar el método utiliza el modificador* ***final****.*  **2.** Ejecuta la aplicación y prueba cómo una pulsación larga sobre un libro nos muestra un menú contextual para compartir, borrar o insertar. |

# 1.5. La barra de acciones (ActionBar)

|  |
| --- |
|  |
| **Vídeo**[**Tutorial**]**:** [*La barra de acciones (ActionBar)*](http://youtu.be/hi4suZHgiE8)*.*  Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones o *ActionBar*. Situada en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones. La barra de acciones aglutina varios elementos; los más habituales son el icono de la aplicación con su nombre y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón *«Overflow»*. Si la aplicación dispone de pestañas (*tabs*), estas podrán situarse en la barra de tareas. También pueden añadirse otros elementos como listas desplegables y otros tipos de widgets incrustados, como el widget de búsqueda que veremos más adelante.  C:\Users\damon\Desktop\20141403021416_s9_image002.jpg  En caso de disponer de menos tamaño el sistema puede redistribuir los elementos y pasar alguna acción al menú de *«Overflow»*. Por ejemplo, en un móvil el ActionBar anterior se podría ver de la siguiente manera:  C:\Users\damon\Desktop\20141403021416_s9_image003.jpg  Los dispositivos anteriores a la 3.0 requerían una tecla física para mostrar el menú de la actividad. Sin embargo, con esta versión dicha tecla deja de ser un requisito de los terminales y los menús pasan a mostrarse en la barra de acciones. En los dispositivos que sí que dispongan de este botón físico, es posible que los tres puntos que representan el menú de Overflow no se representen en la barra de acciones. En este caso tienes que pulsar el botón físico para desplegar este menú.  La barra de acciones se configura igual que los menús disponibles desde la primera versión. Es decir, a través de ficheros XML de menús, almacenados en*res/menu*. Esto permite diseñar el menú de una aplicación de la forma convencional. Cuando la aplicación se ejecute en una versión inferior a la 3.0, el menú se mostrará de forma tradicional. Cuando el usuario pulse la tecla «menú» aparecerá:  C:\Users\damon\Desktop\20141403021416_s9_image004.jpg  Pero, en caso de disponer de una versión 3.0 o superior, se mostrará en la barra de acciones. Más información sobre creación de menú en el siguiente vídeo:  **Vídeo**[**Tutorial**]**:** [*Añadiendo un menú en Android*](http://www.youtube.com/watch?v=UrWMHaJJSnM)*.*  Añadir un *ActionBar* a la aplicación es muy sencillo. Es más, gracias a que se utiliza la misma herramienta que para mostrar menús en dispositivos con versiones anteriores, todas las opciones del *ActionBar* estarán disponibles independientemente de la versión que se esté utilizando. Por lo tanto, no hay que realizar ningún paso para que se visualice la barra de acciones. Todos los temas de Android a partir de la versión 3.0 incorporan por defecto la barra de acciones visible (menos los que acaban en *NoActionBar*). Si se desea ocultar la barra de acciones desde código es muy sencillo; pero mostrarla una vez aplicado un estilo que la deshabilite es imposible.  Desde finales de 2013 también podemos utilizar *ActionBar* en versiones anteriores a la 3.0 gracias a la aparición de una librería de compatibilidad. Veremos cómo realizarlo en uno de los siguientes ejercicios.  **Ejercicio paso a paso*:*** *Añadiendo un ActionBar a nuestra aplicación.*  **1.** Reemplaza el contenido del fichero *res/menu/main.xml* por:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <menu xmlns:android=*"http://schemas.android.com/apk/res/android"* >  <item  android:id=*"@+id/menu\_preferencias"*  android:icon=*"@android:drawable/ic\_menu\_preferences"*  android:orderInCategory=*"95"*  android:showAsAction=*"ifRoom"*  android:title=*"Preferencias"*/>  <item  android:id=*"@+id/menu\_ultimo"*  android:icon=*"@android:drawable/ic\_menu\_set\_as"*  android:orderInCategory=*"100"*  android:showAsAction=*"ifRoom|withText"*  android:title=*"Último leido"*/>  <item  android:id=*"@+id/menu\_buscar"*  android:actionViewClass=*"android.widget.SearchView"*  android:orderInCategory=*"90"*  android:icon=*"@android:drawable/ic\_menu\_search"*  android:showAsAction=*"always|collapseActionView"*  android:title=*"Buscar"*/>  <item  android:id=*"@+id/menu\_acerca"*  android:orderInCategory=*"110"*  android:showAsAction=*"never"*  android:title=*"Acerca de..."*/>  </menu>  Este fichero XML es el que define los iconos y acciones a mostrar. Como ya hemos dicho, para los dispositivos con una versión 3.0 o superior, este menú se muestra en la barra de acciones. Las acciones que indiquen en el atributo showAsAction la palabra always serán mostrados siempre, sin importar si caben o no. El uso de estas acciones debería limitarse a unas pocas, o incluso mejor si no hay ninguna, ya que al forzar que se visualicen todas al mismo tiempo podrían verse incorrectamente. Las acciones que indiquen ifRoom serán mostradas en la barra de acciones si hay espacio disponible, y serán movidas al menú de *Overflow* si no lo hay. En esta categoría se deberían encontrar la mayoría de acciones. Si se indica never la acción nunca se mostrará en la barra de acciones, sin importar el espacio disponible. En este grupo se deberían situar acciones como modificar las preferencias, que deben estar disponibles al usuario; pero no visibles en todo momento.  Las acciones son ordenadas de izquierda a derecha según lo indicado en orderInCategory, con las acciones con un número más pequeño más a la izquierda. Si todas las acciones no caben en la barra, las que tienen un número mayor son movidas al menú de *Overflow*.  **2.** En la actividad principal (*MainActivity.java*), introduce los siguientes métodos:  @Override  **public** **boolean** onCreateOptionsMenu(Menu menu) {  getMenuInflater().inflate(R.menu.*main*, menu);  **return** **super**.onCreateOptionsMenu(menu);  }    @Override  **public** **boolean** onOptionsItemSelected(MenuItem item) {  **switch** (item.getItemId()) {  **case** R.id.*menu\_preferencias*:  Toast.*makeText*(**this**, "Preferencias", Toast.*LENGTH\_LONG*).show();  **break**;  **case** R.id.*menu\_ultimo*:  irUltimoVisitado ();  **break**;  **case** R.id.*menu\_buscar*:  **break**;  **case** R.id.*menu\_acerca*:  AlertDialog.Builder builder = **new** AlertDialog.Builder(**this**);  builder.setMessage("Mensaje de Acerca De");  builder.setPositiveButton(android.R.string.*ok*, **null**);  builder.create().show();  **break**;  }  **return** **false**;  }  El código introducido es idéntico al que se utiliza para introducir un menú en versiones anteriores a la 3.0. Al crear la actividad, el sistema llama al método onCreateOptionsMenu(). En ese método, la actividad infla el fichero XML para crear este menú, y lo devuelve en un parámetro de tipo Menu. La respuesta del método ha de ser true si queremos que se muestre de forma inmediata o false si de momento no queremos mostrarlo.  El segundo método (onOptionsItemSelected()) será llamado cuando se seleccione una opción para que podamos actuar en consecuencia. Este método también es el mismo que ya se utiliza en versiones anteriores a la 3.0.  Tendremos que añadir las funcionalidades que faltan para algunos de los nuevos botones. Como puedes ver, al seleccionar menu\_ultimo, se lanzará el método irUltimoVisitado(). Copia el siguiente método dentro de MainActivity:  **public** **void** irUltimoVisitado() {  SharedPreferences pref = getSharedPreferences("com.example.audiolibros\_internal", *MODE\_PRIVATE*);  **int** id = pref.getInt("ultimo", -1);  **if** (id >= 0) {  mostrarDetalle(id);  } **else** {  Toast.*makeText*(**this**,"Sin última vista",Toast.*LENGTH\_* *LONG*).show();  }  }  Lo que hace este método es recuperar de las preferencias[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftn1) la posición del último libro abierto y lo vuelve a abrir.  Para que este método funcione correctamente, tendremos que guardar antes este valor en las preferencias. Para ello, simplemente copia estas líneas dentro del método mostrarDetalle() para que se ejecuten cada vez que se elija un libro:  SharedPreferences pref = getSharedPreferences("com.example.audiolibros\_internal", *MODE\_PRIVATE*);  SharedPreferences.Editor editor = pref.edit();  editor.putInt("ultimo", id);  editor.commit();  Ejecuta la aplicación; podrás ver como aparece la barra de acciones en la parte de arriba, con los botones que hemos definido.  [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftnref1) Más información sobre las preferencias en [androidcurso.com](http://www.androidcurso.com/index.php/299). |

## 1.5.1. La barra de acciones en versiones anteriores a la 3.0

|  |
| --- |
| ***Nota:*** *Se trata de una sección avanzada que no afecta al resto de ejercicios, ya que este ejercicio se realiza sobre una copia de Audiolibros. Puedes saltarte este apartado y realizarlo más tarde en una primera lectura.*  Desde el año 2013 también podemos añadir una barra de acciones en una versión anterior a la 3.0. Para conseguirlo simplemente tenemos que añadir la librería de compatibilidad android.support.v7.appcompat. En el siguiente ejercicio veremos cómo conseguirlo.  **Ejercicio paso a paso:***Añadiendo un ActionBar en versiones anteriores a la 3.0*  **1.** Ejecuta el proyecto anterior en un terminal o emulador con una versión anterior a la 3.0. Observa como no se muestra la barra de acciones.  **2.** Pulsa el botón *Menú*. Ha de aparecer un menú en la parte inferior:  C:\Users\damon\Desktop\20141403021416_s10_image002.jpg  **3.** Para usar otras librerías de compatibilidad como la v4 o la v13 no tenemos más que copiar el fichero .jar dentro de la carpeta libs de nuestro proyecto. Sin embargo, la librería v7 incorpora recursos por lo que el proceso es algo más costoso  **4.** Selecciono la opción del menú *File / Import /* *Existing Proyect Into Workspace*.  **5.** Pulsa el botón *Browse…* y busca a partir de la carpeta donde has instalado el SDK la ruta *…/extras/android/support/v7/appcompat/*. Pulse en *Finish* para crear un nuevo proyecto con esta librería. En el explorador de paquetes aparecerá con el nombre android-support-v7-appcompat.  **6.** Expande la carpeta /libs y selecciona los dos ficheros .jar que aparecen.  C:\Users\damon\Desktop\20141403021417_s10_image003.jpg  **7.** Sobre estos ficheros selecciona con el botón derecho la opción *Build Path / Add to Build Path*.  **8.** Sobre el nombre del proyecto selecciona con el botón derecho la opción *Build Path / Configure Build Path*. En la pestaña *Order and Export*, selecciona los ficheros .jar que acabas de añadir, para que estén disponibles en los proyectos en los que indiquemos que dependen de este proyecto de librería. Deselecciona *Android Dependencies*. Antes de pulsar *OK* verifica que queda de la siguiente manera:  C:\Users\damon\Desktop\20141403021417_s10_image004.jpg  **9.** Crea una copia del proyecto *AudioLibrosv2* y llámale *AudioLibrosAppCompat*. Para ello, haz clic con el botón derecho en la carpeta de proyecto que aparece en la vista *Package Explorer* y selecciona *Copy*. Entonces, haz clic con el botón derecho en el mismo sitio y selecciona *Paste*.  **10.** En el proyecto *AudioLibrosAppCompat*, con el botón derecho selecciona la opción *Properties* y la categoría *Android*. Pulsa el botón *Add* de la sección *Library* y selecciona el nombre de proyecto que acabamos de crear con la librería. Pulsa OK.  **11.** Elimina el fichero *libs/android-support-v4.jar*. Ahora esta librería es importada desde el proyecto creado anteriormente.  **12.** Abre el fichero *MainActivity.java* y reemplaza la clase que extendemos:  **public** **class** MainActivity **extends** FragmentActivity {  por:  **public** **class** MainActivity **extends** ActionBarActivity {  Apareceran algunos errores, pulsando *Shift-Cltrl-O* desaparecerán.  **13.** Abre *AndroidManifest.xml* y selecciona la pestaña Application. En la sección Application Attributes en el campo *Theme* selecciona *Theme.AppCompat*. Resulta imprescindible utilizar este tema o alguno de sus desdendientes.    C:\Users\damon\Desktop\20141403021417_s10_image005.jpg  **14.** Aún nos queda un último paso. Aunque la forma de procesar las acciones desde el código Java sean idénticas, a la hora de definir un menú desde XML tendremos que realizar un pequeño cambio. El problema aparece porque el atributo showAsAction que usábamos para definir un menú no existía hasta la versión 3.0. Para resolverlo la librería de compatibilidad ha definido un nuevo espacio de nombres para definir este atributo. Abre el fichero *res/menu/main.xml* e introduce un nuevo espacio de nombres en la primera etiqueta del fichero.  xmlns:supportv7="http://schemas.android.com/apk/res-auto"  El alias que hemos escogido para el espacio de nombres es arbitrario; puedes escoger el que quieras.  **15.** Reemplaza los atributos android:showAsAction por supportv7:showAsAction. El resultado final se muestra a continuación:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <menu xmlns:android=*"http://schemas.android.com/apk/res/android"*  **xmlns:supportv7=*"http://schemas.android.com/apk/res-auto"***>  <item  android:id=*"@+id/menu\_preferencias"*  android:icon=*"@android:drawable/ic\_menu\_preferences"*  android:orderInCategory=*"95"*  **supportv7**:showAsAction=*"ifRoom"*  android:title=*"Preferencias"*/>  <item  android:id=*"@+id/menu\_ultimo"*  android:icon=*"@android:drawable/ic\_menu\_set\_as"*  android:orderInCategory=*"100"*  **supportv7**:showAsAction=*"ifRoom|withText"*  android:title=*"Último leido"*/>  <item  android:id=*"@+id/menu\_buscar"*  android:actionViewClass=*"android.widget.SearchView"*  android:orderInCategory=*"90"*  android:icon=*"@android:drawable/ic\_menu\_search"*  **supportv7**:showAsAction=*"always|collapseActionView"*  android:title=*"Buscar"*/>  <item  android:id=*"@+id/menu\_acerca"*  android:orderInCategory=*"110"*  **supportv7**:showAsAction=*"never"*  android:title=*"Acerca de..."*/>  </menu>  **16.** Ejecuta la aplicación en el mismo dispositivo que has usado en el punto 1. El resultado ha de ser similar al siguiente:    C:\Users\damon\Desktop\20141403021417_s10_image006.jpg |

## 1.5.2. Añadiendo pestañas en la barra de acciones

|  |
| --- |
| **Vídeo** [**Tutorial**]**:** [*Anadir pestanas en la barra de acciones*](http://youtu.be/1tLiLlxaVTM)*.*  Android permite introducir pestañas, o tabs, mediante el uso de FragmentTabHost[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftn1). Existe otra alternativa para introducir pestañas en nuestra aplicación; consiste en incorporarlas dentro de la Barra de Acciones. Tiene la ventaja de que, si caben, se integrarán dentro de la Barra de Acciones:  C:\Users\damon\Desktop\20141403021417_s11_image002.jpg  En el proyecto *AudioLibrosv2* usaba el API de la versión 2.2 donde no existía la clase ActionBar. A pesar de esto cuando se ejecutaba en una versión igual o superior a la 3.0, los menús tradicionales eran representados en forma de *Barra de Acciones*. Esto presenta un inconveniente, al no estar esta clase en el API no podemos configurar el ActionBar desde código, y por lo tanto no vamos a poder añadir pestañas. Si queremos añadirlas tenemos dos alternativas, utilizar como versión mínima la 3.0 o añadir a nuestro proyecto la librería de compatibilidad v7.appcompat. En los siguientes dos ejercicio abordaremos la primera solución y plantearemos la segunda solución en una práctica.  **Ejercicio paso a paso:***Pasar Audiolibros de versión mínima 2.2 (API 8) a versión mínima 3.0 (API 11).*  **1.** Crea una copia del proyecto *AudioLibrosv2* y llámale *AudioLibrosv3*. Para ello, haz clic con el botón derecho en la carpeta de proyecto y selecciona *Copy*. Entonces, haz clic con botón derecho en el mismo sitio y selecciona *Paste*.  **2.** La versión mínima para este proyecto ha de ser la 3.0 (nivel de API 11). Asegúrate de que en *AndroidManifest.xml* se indica el siguiente atributo:  <uses-sdk android:minSdkVersion=*"11"*  **3.** Elimina el fichero *libs/android-support-v4.jar*. Ya no tiene sentido añadir esta librería al proyecto.  **4.** En las clases DetalleFragment y SelectorFragment reemplaza:  **import** android.support.v4.app.Fragment;  por:  **import** android.app.Fragment;  **5.** En las clase MainActivity reemplaza:  **import** android.support.v4.app.FragmentActivity;  **import** android.support.v4.app.FragmentTransaction;  por:  **import** android.app.Activity;  **import** android.app.FragmentTransaction;  **6.** Reemplaza también:  **public** **class** MainActivity **extends** FragmentActivity {  por:  **public** **class** MainActivity **extends** Activity {  La clase FragmentActivity pertenece a la librería de compatibilidad y permite crear una actividad con *fragmets*. A partir de la versión 3.0 esto se puede realizar desde cualquier actividad.  **7.** Finalmente reemplaza todas las apariciones del método:  getSupportFragmentManager()  por:  getFragmentManager()  **8.** Verifica que la aplicación funciona.    **Ejercicio paso a paso:***Añadir pestañas en la barra de acciones en la versión 3.0.*  En este ejercicio vamos a introducir en la interfaz de usuario de Audiolibros la posibilidad de filtrar los libros que se muestran en el selector utilizando unas pestañas como las que se muestran a continuación:  C:\Users\damon\Desktop\20141403021417_s11_image005.jpg  **1.** Abre el proyecto *Audiolibrosv3* y añade en el método onCreate() de MainActivity el siguiente código:  // Pestañas  ActionBar actionBar = getActionBar();  actionBar.setNavigationMode(ActionBar.*NAVIGATION\_MODE\_TABS*);  actionBar.addTab(actionBar.newTab()  .setText("Todos")  .setIcon(android.R.drawable.*ic\_menu\_view*)  .setTabListener(**this**));  actionBar.addTab(actionBar.newTab()  .setText("Nuevos")  .setIcon(android.R.drawable.*ic\_menu\_agenda*)  .setTabListener(**this**));  actionBar.addTab(actionBar.newTab()  .setText("Leidos")  .setIcon(android.R.drawable.*ic\_menu\_my\_calendar*)  .setTabListener(**this**));  El método setNavigaitonMode() es necesario para activar la visualización de pestañas. Una barra de acciones puede activar dos modos de navegación: las pestañas y una lista desplegable. En el siguiente apartado estudiaremos otro método alternativo de selección basado en una lista desplegable.  Las siguientes tres sentencias permiten insertar tres pestañas. Aparecerán en el orden en que se añaden usando el método addTab(). El método newTab() crea la nueva pestaña, que es configurada a continuación, asignándole un texto, un icono y un escuchador.  **2.** Pulsa *Shifth-Ctrl-O*. Te preguntará de dónde quieres importar la clase ActionBar. Selecciona android.app.ActionBar. Recuerda que en este proyecto ya no tienes que utilizar la librería de compatibilidad.  **3.** Hemos asignado nuestra clase como escuchador de las tres etiquetas. Por lo tanto, nuestra clase tiene que implementar la interfaz TabListener:  **Public** **class** MainActivity **extends** Activity **implements** TabListener  **4.** Tras el paso anterior la palabra MainActivity aparecerá subrayada en rojo, indicándonos que todavía no hemos implementado los métodos de la interfaz. Sitúa el cursor del ratón sobre esta palabra y selecciona la opción *Add unimplimented methods*. Se insertarán al final de la clase tres nuevos métodos. Los nombres de los métodos nos indican claramente cuándo van a ser llamados. Los tres tienen dos parámetros, la etiqueta que produce el evento (Tab) y un manejador de *fragment* (FragmentTransaction) para que podamos manipular los *fragments* de la actividad. Aunque no es el caso de nuestro ejemplo, en la mayoría de las ocasiones las pestañas se usan para asignar a cada una de ellas un *fragment*.  **5.** Tal y como se muestra a continuación de los tres métodos solo introducimos código en uno:  @Override  **public** **void** onTabSelected(Tab tab, FragmentTransaction ft) {  **switch** (tab.getPosition()) {  **case** 0: //Todos  adaptador.setNovedad(**false**);  adaptador.setLeido(**false**);  **break**;  **case** 1: //Nuevos  adaptador.setNovedad(**true**);  adaptador.setLeido(**false**);  **break**;  **case** 2: //Leidos  adaptador.setNovedad(**false**);  adaptador.setLeido(**true**);  **break**;  }  adaptador.notifyDataSetChanged();  }    @Override  **public** **void** onTabReselected(Tab tab, FragmentTransaction ft) {  }    @Override  **public** **void** onTabUnselected(Tab tab, FragmentTransaction ft) {  }  Según la posición de la etiqueta seleccionada, modificamos la configuración del adaptador para que visualice todos los libros, las novedades y los leídos.  Terminamos notificando al adaptador que los datos han cambiado.  **6.** El siguiente paso va a ser modificar nuestro adaptador para que permita filtrar la información que mostrará al usuario. Para ello vamos a crear una nueva clase, descendiente de AdaptadorLibros, que incorpore la funcionalidad adicional. Pulsa con el botón derecho sobre la clase com.example.audiolibros del proyecto y selecciona *New / Class*. El nombre de la nueva clase va a ser AdaptadorLibrosFiltro y su código se muestra a continuación:  **public** **class** AdaptadorLibrosFiltro **extends** AdaptadorLibros {    **private** Vector<Libro> vectorSinFiltro;// Vector con todos los libros  **private** Vector<Integer> indiceFiltro; // Índice en vectorSinFiltro de  // Cada elemento de vectorLibros  **private** String busqueda = ""; // Búsqueda sobre autor o título  **private** String genero = ""; // Género seleccionado  **private** **boolean** novedad = **false**; // Si queremos ver solo novedades  **private** **boolean** leido = **false**; // Si queremos ver solo leidos    **public** AdaptadorLibrosFiltro(Context contexto, Vector<Libro> vectorLibros) {  **super**(contexto, vectorLibros);  vectorSinFiltro = vectorLibros;  recalculaFiltro();  }    **public** **void** setBusqueda(String busqueda) {  **this**.busqueda = busqueda.toLowerCase();  recalculaFiltro();  }    **public** **void** setGenero(String genero) {  **this**.genero = genero;  recalculaFiltro();  }    **public** **void** setNovedad(**boolean** novedad) {  **this**.novedad = novedad;  recalculaFiltro();  }    **public** **void** setLeido(**boolean** leido) {  **this**.leido = leido;  recalculaFiltro();  }    **public** **void** recalculaFiltro() {  vectorLibros = **new** Vector<Libro>();  indiceFiltro = **new** Vector<Integer>();  **for** (**int** i = 0; i < vectorSinFiltro.size(); i++) {  Libro libro = vectorSinFiltro.elementAt(i);  **if** ((libro.titulo.toLowerCase().contains(busqueda) ||  libro.autor.toLowerCase().contains(busqueda))  && (libro.genero.startsWith(genero))  && (!novedad || (novedad && libro.novedad))  && (!leido || (leido && libro.leido))) {  vectorLibros.add(libro);  indiceFiltro.add(i);  }  }  }    **public** **long** getItemId(**int** posicion) {  **return** indiceFiltro.elementAt(posicion);  }    **public** **void** borrar(**int** posicion){  vectorSinFiltro.remove((**int**)getItemId(posicion));  recalculaFiltro();  }    **public** **void** insertar(Libro libro){  vectorSinFiltro.add(libro);  recalculaFiltro();  }  }  Para estudiar el funcionamiento de esta clase es recomendable que repases primero la clase padre AdaptadorLibros. Observa como la variable vectorLibros ha sido marcada con el modificador protected lo que nos permite acceder a ella desde un descendiente. Esta variable va a representar el subconjunto de libros que estamos visualizando a través del adaptador. La gran diferencia con respecto a la versión anterior es que ahora dispondremos de un nuevo vector, vectorSinFiltro, con todos los libros disponibles. El vector indiceFiltro es utilizado para relacionar estos dos vectores. El número de elementos de indiceFiltro es igual al de vectorLibros. Y en cada posición se almacena el índice de este mismo libro, pero en vectorSinFiltro.  Dispondremos de cuatro posibles filtros de selección: búsqueda en autor y título, género literario, es una novedad y leído por el usuario. Se han creado cuatro variables para almacenar los filtros de selección busqueda, genero, novedad y leido.  Observa cómo tras el constructor se han añadido cuatro métodos *setter* para modificar el criterio de búsqueda. Al final del constructor y de cada uno de estos *setters* se llama a recalculaFiltro(). La finalidad de este método es calcular un nuevo vectorLibros, y su correspondiente indiceFiltro, según el nuevo criterio de búsqueda.  De los cuatro métodos que ha de incluir un BaseAdapter, solo es necesario modificar getItemId(). Es importante que tengas claro que en la nueva clase las variables posicion e id representan cosas diferentes: posicion hace referencia a los elementos visualizados y corresponden a índices de vectorLibros. Mientras que id hace referencia a un libro concreto y corresponden a índices de vectorSinFiltro. El método getItemId() devuelve el id de una determinada posición utilizando la información guardada en indiceFiltro.  Con el adaptador anterior, cuando queríamos borrar un elemento o insertar uno nuevo modificábamos directamente vectorLibros (ver menú contextual en SelectorFragment). Ahora tendremos que modificar el vector vectorSinFiltro y recalcular vectorLibros. Para realizar estas tareas se han añadido dos nuevos métodos al final de la clase.  **7.** Abre la clase SelectorFragment y modifica las siguientes tres líneas:  **private** AdaptadorLibros adaptador;  por:  **private** AdaptadorLibrosFiltro adaptador;    vectorLibros.remove((**int**) id);  por:  adaptador.borrar((**int**) id);    vectorLibros.add(vectorLibros.elementAt((**int**) id));  por:  adaptador.insertar((Libro) adaptador.getItem(posicion));  **8.** Abre la clase Aplicacion y reemplaza las tres apariciones de AdaptadorLibros por AdaptadorLibrosFiltro.  **9.** Añade en la clase MainActivity la siguiente variable:  **private** AdaptadorLibrosFiltro adaptador;  **10.** Dentro del método onCreate() añade:  adaptador = ((Aplicacion) getApplicationContext()).getAdaptador();  **11.** Ejecuta el programa y verifica que según la pestaña seleccionada se visualiza una lista de libros diferente.  **Ejercicio paso a paso:** *Añadir pestañas en la barra de acciones en la versión anterior a la 3.0.*  ***Nota:*** *Se trata de un ejercicio avanzado que no afecta al resto.*  **1.** Abre el proyecto *AudiolibrosAppCompact* y realiza el punto 1 del ejercicio anterior. Pero ahora, reemplaza getActionBar() por getSupportActionBar().  **2.** Pulsa *Shifth-Ctrl-O*. Te preguntará de dónde quieres importar la clase ActionBar. Selecciona ahora android.supportv7.app.ActionBar. Recuerda que en este proyecto tienes que utilizar la librería de compatibilidad.  **3.** Nuestra actividad tiene que implementar la interfaz TabListener:  **Public** **class** MainActivity **extends** ActionBarActivity **implements** TabListener  Pulsa *Shifth-Ctrl-O* e indica android.supportv7.app.ActionBar.TabListener.  **4.** El resto de los pasos son iguales al ejercicio anterior.    [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftnref1)<http://androidcurso.com/index.php/452> |

## 1.5.3. Añadiendo preferencias de usuario mediante PreferenceFragment

|  |  |  |
| --- | --- | --- |
| Desde la versión 3.0 de Android, la clase PreferenceActivity está obsoleta. Aunque se puede seguir utilizando, es preferible utilizar PreferenceFragment, la cual tiene la misma funcionalidad, pero a través de un *fragment*. Esta clase solo está disponible desde la versión 3.0 y, por lo tanto, no es compatible con versiones anteriores. Desgraciadamente, tampoco ha sido incluida en la librería de compatibilidad. Por lo tanto, en versiones anteriores a la 3.0 tendrás que utilizar necesariamente PreferenceActivity.  **Ejercicio paso a paso:** Añadiendo preferencias de usuario mediante PreferenceFragment.  **1.** Abre el proyecto *AudioLibrosv3*.  **2.** Crea una nueva clase llamada *PreferenciasFragment* (dentro del paquete de *com.example.audiolibros.fragments*) con el siguiente código:  **public** **class** PreferenciasFragment **extends** PreferenceFragment {  @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  addPreferencesFromResource(R.xml.*preferences*);  }  }  Pulsa *Shifth-Ctrl-O* y selecciona com.example.audiolibros.R. Como puedes ver, esta clase se utiliza exactamente igual que ActivityFragment.  **3.** Crea un nuevo fichero XML llamado *preferences.xml* dentro de *res/xml*:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <PreferenceScreen xmlns:android=*"http://schemas.android.com/apk/res/android"*>  <CheckBoxPreference  android:key=*"pref\_autoreproducir"*  android:title=*"Reproducir automáticamente"*  android:summary=*"El audio se inicia automáticamente al abrirlo"*  android:defaultValue=*"true"* />  </PreferenceScreen>  **4.** Adicionalmente, tendremos que crear una actividad que muestre este *fragment*. Para ello, crea una nueva clase con el siguiente código:  **public** **class** PreferenciasActivity **extends** Activity {  @Override  **protected** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  getFragmentManager().beginTransaction()  .replace(android.R.id.*content*, **new** PreferenciasFragment()).commit();  }  }  Este código simplemente coloca el *fragment* como único contenido de la actividad.  **5.** Recuerda registrar esta actividad añadiendo en *AndroidManifest*:  <activity android:name=*"PreferenciasActivity"*/>  **6.** Por último, tendremos que lanzar esta actividad cuando el usuario la pida. Para ello, en MainActivity dentro de onOptionsItemSelected(), si el elemento elegido es menu\_preferencias, lanza la nueva actividad:  **case** R.id.menu\_preferencias:  Intent i = **new** Intent(**this**, PreferenciasActivity.**class**);  startActivity(i);  **break**;  **Práctica:** *Añadir dinámicamente PreferenciasFragment.*  El objetivo de esta práctica es conseguir que el *fragment* realizado en el ejercicio anterior, PreferenciasFragment, se visualice conjuntamente con otros *fragments*.  **1.** En caso de estar en una pantalla pequeña, simplemente intercambia el *fragment* que esté visible por PreferenciasFragment, permitiendo al usuario volver hacia atrás al pulsar la tecla «atrás».  **2.** En el caso de estar en una pantalla grande, tendrás que intercambiar el *fragment* SelectorFragment por PreferenceFragment, dejando visible el *fragment* DetalleFragment para que el usuario aún pueda escuchar el libro mientras modifica las preferencias.  **3.** Finalmente, añade la funcionalidad de la opción de preferencias «Reproducir automáticamente». Si está seleccionado, al abrir un libro el audio asociado a él se reproducirá en cuanto esté disponible. Si se deja sin seleccionar, el usuario tendrá que darle al *play* manualmente.  ***Nota:*** *Recuerda que si añadimos un fragment desde XML, después no podemos hacer transiciones con él. En la vista para tabletas, el fragment está añadido desde XML, por lo que tendrás que cambiarlo a código.*   |  |  | | --- | --- | |  | **Solución:** *Añadir dinámicamente PreferenciasFragment***.** |   **1.** Modifica el fichero layout-large/activity\_main.xml para que en su izquierda tenga un LinearLayout en lugar de un fragment estático. Será utilizado como contenedor para insertar el fragment que nos interese en cada momento.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="horizontal" >  **<LinearLayout**  **android:id="@+id/contenedor\_izquierdo"**  **android:orientation="vertical"**  **android:layout\_width="0dp"**  **android:layout\_height="match\_parent"**  **android:layout\_weight="1" />**  <fragment  android:id="@+id/detalle\_fragment"  android:name="com.example.audiolibros.fragments.DetalleFragment"  android:layout\_width="0dp"  android:layout\_height="match\_parent"  android:layout\_weight="2" />  </LinearLayout>  **2.** Cuando se inicie la actividad tendremos que poner un SelectorFragment en este contenedor cuando estemos en pantalla grande. Pero si estamos en pantalla pequeña, lo tendremos que insertar en el contenedor contenedor\_pequeno. Para ello, en el método onCreate de MainActivity, elimina el if donde se insertaba un fragment y reemplázalo por las siguientes líneas:  **int** idContenedor = (findViewById(R.id.*contenedor\_pequeno*) != **null**) ?  R.id.*contenedor\_pequeno* : R.id.*contenedor\_izquierdo*;  SelectorFragment primerFragment = **new** SelectorFragment();  getFragmentManager().beginTransaction().add(idContenedor, primerFragment).commit();  **3.** Crea el siguiente método en MainActivity. Su finalidad es reemplazar el *fragment* actual por un nuevo *fragment* de preferencias. El contenedor donde se reemplaza puede ser contenedor\_pequeno o contenedor\_izquierdo según el tamaño de pantalla.  **public** **void** abrePreferencias() {  **int** idContenedor = (findViewById(R.id.*contenedor\_pequeno*) != **null**) ?  R.id.*contenedor\_pequeno* : R.id.*contenedor\_izquierdo*;  PreferenciasFragment prefFragment = **new** PreferenciasFragment();  getFragmentManager().beginTransaction()  .replace(idContenedor, prefFragment)  .addToBackStack(**null**)  .commit();  }  **4.** En el método onOptionsItemSelected() llama a este método cuando se seleccione la opción adecuada:  **case** R.id.menu\_preferencias:  abrePreferencias();  **break**;  **5.** Para añadir la funcionalidad a la preferencia autorreproducir, añade en el método onPrepared() de la clase DetalleFragment el código:  SharedPreferences preferencias = PreferenceManager.*getDefaultSharedPreferences*(getActivity());  **if** (preferencias.getBoolean("pref\_autoreproducir", **true**)) {  mediaPlayer.start();  }  **6.** Verifica que todo funcione correctamente. |

# 1.6. Servicios de búsquedas

|  |
| --- |
|  |

|  |
| --- |
|  |
| Una de las características que ofrece Android a partir de la versión 3.0 es la posibilidad de realizar búsquedas a través de la vista SearchView. Este elemento puede colocarse tanto dentro de un *layout* como en la barra de acciones. En esta sección veremos cómo añadirlo en la barra de acciones. Es muy similar a incorporarlo dentro de un *layout*.  El *framework* de búsquedas nos ofrece las siguientes posibilidades:   Proporcionar sugerencias de búsqueda basadas en consultas recientes.   Proporcionar sugerencias de búsqueda personalizadas que responden a los datos disponibles en nuestra aplicación.   Podemos habilitar la búsqueda por voz.    Sin embargo, no nos proporciona ningún método para realizar las búsquedas en nuestra estructura de datos; tendremos que hacerlo nosotros. Por ejemplo, si estamos usando una base de datos, tendremos que hacer la consulta nosotros mismos.  **Ejercicio paso a paso:***Incorporando búsquedas en nuestra aplicación.*  **1.** La versión mínima para usar el servicio de búsqueda ha de ser la 3.0 (nivel de API 11). Puedes partir del proyecto *AudioLibrosv3*.  **2.** Para incorporar una vista de búsquedas en un ActionBar añade el siguiente elemento dentro del fichero XML de menú. Nosotros ya lo añadimos dentro de *res/menu/main.xml*.  <item  android:id=*"@+id/menu\_buscar"*  android:actionViewClass=*"android.widget.SearchView"*  android:icon=*"@android:drawable/ic\_menu\_search"*  android:orderInCategory=*"90"*  android:showAsAction=*"ifRoom|collapseActionView"*  android:title=*"Buscar"*/>  Observa el atributo actionViewClass que nos permite asociar una *actionView* a una opción del menú. Una *actionView* es una vista que aparece en la barra de acción cuando el usuario selecciona una determinada acción. En este ejemplo queremos que se visualice una SearchView al seleccionar esta opción.  **3.** El segundo paso será asociar este nuevo elemento con el servicio que recibirá las búsquedas. Para ello, copia el siguiente código dentro del método onCreateOptionsMenu(), después de inflar el archivo XML del menú, pero antes de retornar del método:  SearchManager searchManager = (SearchManager) getSystemService(Context.*SEARCH\_SERVICE*);  SearchView searchView = (SearchView) menu.findItem(R.id.*menu\_buscar*).getActionView();  searchView.setSearchableInfo(searchManager.getSearchableInfo(getComponentName()));  **4.** Después, tendremos que definir en *AndroidManifest.xml* el servicio de búsquedas y decirle a qué actividad tendrá que enviar estas búsquedas, en forma de intenciones. En nuestro caso, la actividad que recibirá las búsquedas será nuestra actividad principal. Para ello, copia el siguiente código dentro del elemento que describe la actividad principal:  <activity  android:name=*"com.example.audiolibros.MainActivity"*  android:label=*"@string/app\_name"*  **android:launchMode="*singleTop"*>**  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  **<intent-filter>**  **<action android:name=*"android.intent.action.SEARCH"* />**  **</intent-filter>**  **<meta-data**  **android:name=*"android.app.searchable"***  **android:resource=*"@xml/searchable"* />**  </activity>  Como puedes ver, estamos indicando que nuestra actividad puede procesar intenciones con la acción SEARCH. También indicamos que este servicio tiene un fichero XML asociado, el cual describe las propiedades del cuadro de búsquedas. Crea un nuevo fichero XML dentro de *res/xml* llamado *searchable.xml* con el siguiente contenido:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <searchable xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:hint=*"Buscar libro"*  android:label=*"@string/app\_name"*>  </searchable>  **5.** El servicio va a enviar las búsquedas en forma de intenciones a la actividad MainActivity. Si no indicamos nada, cada búsqueda lanzará una nueva instancia de esta actividad. Para evitar esto, hemos añadido en el paso 4 el siguiente atributo a las propiedades de la actividad MainActivity (junto a las propiedades *label* y *name*):  android:launchMode=*"singleTop"*  Esto evitará que la actividad principal sea creada de nuevo al recibir una búsqueda.  **6.** Por último, tendremos que capturar los nuevos lanzamientos de la actividad y ver si se trata de una intención de búsqueda. En tal caso indicamos al adaptador la búsqueda que nos han pasado en la intención:  **protected** **void** onNewIntent(Intent intent) {  **if** (intent.getAction() != **null**) {  **if** (intent.getAction().equals(Intent.*ACTION\_SEARCH*)) {  adaptador.setBusqueda(intent.getStringExtra(SearchManager.*QUERY*));  adaptador.notifyDataSetChanged();  }  }  }  **7.** Verifica que todo funcione correctamente. |

# 1.7. Navigation Drawer

|  |
| --- |
|  |

|  |
| --- |
|  |
| En el Google IO del 2013 se presentó una de las últimas incorporaciones a la interfaz de usuario en Android: Navigation Drawer, que podríamos traducir al castellano como ventana de navegación lateral deslizable. A pesar de no formar parte del API hasta hace muy poco tiempo, seguramente lo habrás utilizado con anterioridad en aplicaciones que utilizaban librerías externas.  C:\Users\damon\Desktop\20141403021417_s14_image001.jpg  El usuario sabrá que dispone de un Navigation Drawer cuando aparecen tres pequeñas rayas a la izquierda de la barra de acciones. Si pulsa sobre este icono o mediante un desplazamiento horizontal desde el lado izquierdo de la pantalla hacia la derecha, se mostrará el *Navigation Drawer*. Este aparecerá deslizándose de izquierda a derecha, sin llegar a ocupar la totalidad de la pantalla.  Aunque suele tener una estructura de menú, es posible incorporar en él cualquier tipo de elemento. Como veremos más adelante, vamos a poder definir cualquier *layout* y asociarlo al *Navigation Drawer*.  Implementar un *Navigation Drawer* es algo complejo. Para una mejor comprensión comenzamos enumerando las clases a utilizar:  DrawerLayout: Nuevo *layout* proporcionado por la librería de compatibilidad, que usaremos como contenedor raíz del *layout* de la actividad.  ActionBarDrawerToggle: Hace de nexo entre un DrawerLayout y la Barra de acciones. Además, permite implementar la funcionalidad del *Navigation Drawer*.  DrawerListener: Interfaz que permite reaccionar ante los eventos de abertura y cierre del *Navigation Drawer*. La clase ActionBarDrawerToggle dispone de esta interfaz.  Los pasos para incorporar un *Navigation Drawer* son los siguientes:  1. Crea un *layout* para la actividad. Veamos un ejemplo:  <android.support.v4.widget.DrawerLayout  xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/drawer\_layout"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <!--El contenido de la actividad -->  <LinearLayout   android:id="@+id/contenedor\_pequeno"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="horizontal"/>  <!--El contenido del Navigation Drawer -->  <ListView   android:id="@+id/left\_drawer"  android:layout\_width="240dp"  android:layout\_height="match\_parent"  android:layout\_gravity="start"  android:choiceMode="singleChoice"  android:divider="@android:color/transparent"  android:dividerHeight="0dp"  android:background="#111"/> </android.support.v4.widget.DrawerLayout>  Observa como utilizamos como raíz la etiqueta <DrawerLayout> de la librería de compatibilidad v4. Ha de contener dos elementos. El primero será la vista que mostrará la actividad. En este ejemplo se insertará posteriormente un Fragment. El segundo es la vista que mostrará el *Navigation Drawer*. En este ejemplo, se ha escogido un ListView para mostrar un menú de opciones. Google nos recomienda un ancho mínimo de 240 dp y máximo 320 dp. De esta manera el usuario siempre verá parte del Layout principal. Por medio del atributo layout\_gravity podemos indicar dónde aparece el *Navigation Drawer*. Lo habitual es utilizar los valores start (o el equivalente left en idiomas que se escriben de izquierda a derecha), aunque también se puede utilizar end (o rigth).  2. Crea la funcionalidad asociada a la vista del *Navigation Drawer*. En caso de basarse en un ListView tendremos que introducir los elementos y escribir el escuchador de eventos para cuando se seleccione un elemento de la vista. En caso de usar otro tipo de vistas, las acciones a realizar serían muy diferentes.  3. Asocia el *Navigation Drawer* a la barra de acciones y controla la apertura y cierre con el icono de la aplicación. Puedes utilizar el siguiente código:  public class MainActivity extends Activity {  private DrawerLayout drawerLayout;  private ActionBarDrawerToggle drawerToggle;  ...   public void onCreate(Bundle savedInstanceState) {  ...  // Navigation Drawer  drawerLayout = (DrawerLayout) findViewById(R.id.drawer\_layout);  drawerToggle = new ActionBarDrawerToggle(  this, // Actividad que nos contiene  drawerLayout, // DrawerLayout   R.drawable.ic\_navigation\_drawer, // Icono del Nav. Drawer  R.string.drawer\_open, // Descripcion abrimos Nav. Drawer  R.string.drawer\_close) // Descripcion cierre Nav. Drawer  {  public void onDrawerClosed(View view) {  getActionBar().setTitle(getTitle()); //Titulo App  }   public void onDrawerOpened(View drawerView) {  getActionBar().setTitle("Seleccione opción");  }  };  drawerLayout.setDrawerListener(drawerToggle);  getActionBar().setDisplayHomeAsUpEnabled(true);  }   @Override  protected void onPostCreate(Bundle savedInstanceState) {  super.onPostCreate(savedInstanceState);  drawerToggle.syncState();  }   @Override  public void onConfigurationChanged(Configuration newConfig) {  super.onConfigurationChanged(newConfig);  drawerToggle.onConfigurationChanged(newConfig);  }   @Override  public boolean onOptionsItemSelected(MenuItem item) {  if (drawerToggle.onOptionsItemSelected(item)) {  return true;  }  ...   }  ... }  Comenzamos añadiendo un par de variables globales. En el método onCreate() está el código necesario para inicializarlas. drawerLayout corresponde con el *layout* de la actividad. Para la creación de drawerLayout se utiliza un constructor con cinco parámetros: el contexto, el DrawerLayout que asociamos, el icono característico y dos recursos de *strings* con que describen en qué estado se encuentra. Esta descripción puede ser utilizada para mejorar la accesibilidad del sistema, aunque en la mayoría de los sistemas no se representa. Añadimos a continuación dos métodos *callback*. El primero se llama cuando se cierra el *Navigation Drawer* y el segundo, cuando se abre. Las acciones asociadas consistirán en cambiar el título de la barra de acciones. En el primer caso, dejamos el título de la aplicación y en el segundo, un texto para indicar al usuario que ha de seleccionar una opción. En la siguiente línea se indica que los eventos de apertura y cierre de drawerLayout sean enviados a drawerToggle. Finalmente indicamos a la barra de acciones que muestre el icono y active el *Navigation Drawer.* Cuando quieras desactivarlo utiliza como parámetro false.  A continuación sobrescribimos un par de métodos *callback* de la actividad. Con el primero conseguimos que tras crear la actividad, el drawerToggle sea sincronizado. Con el segundo conseguimos que cuando se reciba el evento de cambio de configuración en la actividad, este sea propagado al drawerToggle.  Finalmente tenemos que añadir código al método onOptionsItemSelected() que es llamado cada vez que se selecciona una opción de la barra de acciones. Lo primero que hay que hacer es llamar al onOptionsItemSelected() para ver si se ha seleccionado el icono del *Navigation Drawer*. En tal caso, terminamos devolviendo true para indicar que la pulsación ya ha sido tratada. En caso contrario tendrás que añadir el código necesario para detectar otras acciones de la barra de acciones.  4. Para terminar va a ser imprescindible definir los recursos usados en el punto anterior. Abre el fichero *res/values/strings.xml* y añade estas líneas:  <string name=*"drawer\_open"*>navigation drawer abierto</string>  <string name=*"drawer\_close"*>navigation drawer cerrado</string>  También hemos utilizado el recurso R.drawable.ic\_drawer, que queremos que corresponda al icono característico del *Navigation Drawer*. Este icono no se encuentra en los recursos del sistema. Este elemento de navegación ha sido creado muy recientemente y no existía en la versión 3.0 de Android. Para obtener este icono en las diferentes resoluciones puedes acceder a la herramienta Web [Android Asset Studio](http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html) en la sección [Navigation Drawer Indicator Generator](http://android-ui-utils.googlecode.com/hg/asset-studio/dist/icons-nav-drawer-indicator.html#theme=light&color=33b5e5%2C40)[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftn1). Pulsa en el botón *Download ZIP* para obtener el icono en diferentes resoluciones y copia cada uno de los iconos en la carpeta *drawable* correspondiente.  **Ejercicio paso a paso:***Añadir un Navigation Drawer en Audiolibros.*    En este ejercicio vamos a aplicar lo explicado en los puntos anteriores al proyecto desarrollado en este capítulo. La funcionalidad que queremos conseguir es usar el *Navigation Drawer* para seleccionar el género de libros a visualizar, de manera que en el *fragment* de selección solo aparezcan los libros del género literario seleccionado.  C:\Users\damon\Desktop\20141403021417_s14_image004.jpg  **1.** Abre el proyecto Audiolibrosv3.  ***Nota:*** *Navigation Drawer ha sido incorporado en la librería de compatibilidad, por lo tanto puedes usarlo también en versiones anteriores a la 3.*  **2.** Verifica en *Android SDK Manager* que está instalada una versión de *Android Support Library* igual o superior a la 18:  C:\Users\damon\Desktop\20141403021417_s14_image005.jpg  **3.** Aseguraté de que en tu proyecto se ha incluido la librería de compatibilidad v4.  C:\Users\damon\Desktop\20141403021417_s14_image006.jpg  **4.** Reemplaza el código del *layout* *activity\_main.xml* por el mostrado en el punto 1 de la explicación anterior.  **5.** Modifica también *layout-large/activity\_main.xml* de manera que todo el contenido anterior sea incorporado dentro de la etiqueta DrawerLayout en sustitución del LinearLayout.  **6.** Realiza los cambios propuestos en el punto 3 y 4 de la explicación anterior.  **7.** Para crear la funcionalidad del *Navigation Drawer* vamos a utilizar un ListView. Primero declaramos dos variables globales de MainActivity.  **private** ListView listaMenu; // Opciones del Navigation Drawer  y lo configuramos en el método onCreate() con el siguiente código:  listaMenu = (ListView) findViewById(R.id.*left\_drawer*);  listaMenu.setAdapter(**new** ArrayAdapter<String>(**this**, R.layout.*drawer\_list\_item*, Libro.*G\_ARRAY*));  listaMenu.setItemChecked(0, **true**);  listaMenu.setOnItemClickListener(**new** ListView.OnItemClickListener() {  @Override  **public** **void** onItemClick(AdapterView parent, View vista, **int** pos, **long** id) {  **if** (pos == 0) {  adaptador.setGenero("");  } **else** {  adaptador.setGenero(Libro.*G\_ARRAY*[pos]);  }  adaptador.notifyDataSetChanged();  drawerLayout.closeDrawer(listaMenu);  }  });  Este ListView ha sido insertado en el *layout* de la actividad, por lo que podemos buscarlo por su *id*. A continuación le asociamos el adaptador ArrayAdapter<String> que nos permite introducirle la lista de ítems a partir de un array de *Strings*. El constructor de este adaptador requiere tres parámetros: el contexto, un *layout* con el elemento a visualizar en el ListView (será creado a continuación) y un array de *String* con los géneros literarios (creado en la clase Libro). La siguiente línea hace que el primer elemento de la lista se muestre como marcado. Como puedes ver en la clase Libro corresponde a «Todos los géneros» y queremos que al principio se muestre seleccionado.  Finalmente creamos un nuevo objeto anónimo de la interfaz OnItemClickListener que ejecutará el método onItemClick() cuando el usuario seleccione un elemento de la lista. Con esta selección el usuario podrá configurar el objeto adaptador que es el que controla los libros que se visualizan en el GridView del selector de libros. Si el usuario selecciona la primera opción, «Todos los géneros», se llama al método setGenero() sin indicar ningún género en concreto. Si se selecciona otra posición se llamará a setGenero() con el género seleccionado según el array de géneros definido en Libro.  Terminamos notificando a adaptador que su conjunto de datos ha cambiado, y cerrando el *Navigation Drawer*, indicando la vista correspondiente al *Navigation Drawer*.  **8.** En el ListView que acabamos de crear hemos indicado que cada elemento de la lista se define en R.layout.drawer\_list\_item. Crea este nuevo *layout* a partir del siguiente código.  <TextView xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:id=*"@android:id/text1"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:gravity=*"center\_vertical"*  android:paddingLeft=*"16dp"*  android:paddingRight=*"16dp"*  android:textColor=*"#fff"*  android:textAppearance=*"?android:attr/textAppearanceMedium"*  android:background=*"?android:attr/activatedBackgroundIndicator"*/>  Si actualizas el nivel mínimo de API a 14 (versión 4.0). Sería interesante que usaras estos atributos para adaptarte a la configuración estándar de Android.  android:textAppearance=*"?android:attr/textAppearanceListItemSmall"*  android:minHeight=*"?android:attr/listPreferredItemHeightSmall"*  **9.** Verifica el funcionamiento del programa.    **Práctica:** *Mostrar el género seleccionado en el título.*  Podrías modificar la aplicación para que en el caso de que el usuario seleccione un género literario, este sea mostrado en el título. Si no se selecciona nada, o todos los géneros, ha de mostrarse «Audiolibros».  **Práctica:** *Añadir botón en Navigation Drawer.*  En el ejercicio anterior hemos creado un Navigation Drawer utilizando la vista ListView. Aunque suele ser la alternativa más habitual podemos utilizar cualquier tipo de vista o incluso un *layout* que combine varias vistas.  **1.** Modifica los layouts adecuados para que la vista asociada al Navigation Drawable sea un LinearLayout que contenga dos elementos, un ImageButton y el ListView de selección de género. La imagen asociada al botón ha de ser @android:drawable/ic\_menu\_preferences. La apariencia ha de ser similar a la siguiente:    C:\Users\damon\Desktop\20141403021417_s14_image009.jpg  ***Nota:*** *Recuerda que has de modificarlo para el tamaño pequeño y grande .*  **2.** Observa cómo hemos escrito drawerLayout.closeDrawer(listaMenu) para cerrar el *Navigation Drawer.* Ahora, listaMenu ya no es la vista que queremos cerrar. Tendrás que crear una variable global que contenga la nueva vista y utilizarla en este método.  **3.** Verifica si funciona hasta este punto.  **4.** Asocia un evento onClick al nuevo botón, de manera que se cierre el *Navigation Drawer* y luego se llame a abrePreferencias().  **5.** Verifica que cuando se pulsa el botón se abre el *fragment* de preferencias.   |  |  | | --- | --- | |  | **Solución:** *Añadir botón en Navigation Drawer***.** |   **1.** Crea un nuevo *layout* con nombre navigation\_drawer con el siguiente código:  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/vistaDrawer"  android:layout\_width="240dp"  android:layout\_height="match\_parent"  android:layout\_gravity="start"  android:orientation="vertical" >  <ImageButton  android:layout\_width="240dp"  android:layout\_height="wrap\_content"  android:onClick="clickBoton"  android:src="@android:drawable/ic\_menu\_preferences" />  <ListView  android:id="@+id/left\_drawer"  android:layout\_width="240dp"  android:layout\_height="match\_parent"  android:background="#111"  android:choiceMode="singleChoice"  android:divider="@android:color/transparent"  android:dividerHeight="0dp" />  </LinearLayout>  Modifica los ficheros *res/layout/activity\_main.xml* y *res/layout-large/activity\_main.xml* para que se incluya el *layout* anterior en el *Navigation Drawable*, tal y como se indica a continuación:  <android.support.v4.widget.DrawerLayout … >  <!-- El contenido de la actividad -->  …  <!-- El contenido del Navigation Drawer -->  <include layout=*"@layout/navigation\_drawer"*/>  </android.support.v4.widget.DrawerLayout>  Crea la siguiente variable global en MainActivity:  **private** View vistaDrawer;  En el método onCreate() incluye:  vistaDrawer = findViewById(R.id.*vistaDrawer*);  En este mismo método busca la siguiente línea:  drawerLayout.closeDrawer(listaMenu);  y reemplázala por:  drawerLayout.closeDrawer(vistaDrawer);  Añade el siguiente método:  **public** **void** clickBoton(View view) {  drawerLayout.closeDrawer(vistaDrawer);  abrePreferencias();  }  [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftnref1) <http://android-ui-utils.googlecode.com/hg/asset-studio/dist/icons-nav-drawer-indicator.html> |

# 1.8. Animaciones

|  |
| --- |
|  |
| Android nos proporciona tres mecanismos para crear animaciones:  Animaciones de Drawables: La clase AnimationDrawable permite crear *drawables* que reproducen una animación fotograma a fotograma.   Animaciones de vistas: También conocidas como animaciones *tween*, permiten crear efectos de translación, rotación, zum y alfa a cualquier vista de nuestra aplicación.   Animaciones de propiedades: Nuevo mecanismo incorporado en Android 3.0. Permite animar cualquier propiedad de cualquier objeto, sea una vista o no. Además modifica el objeto en sí, no solamente cambia su representación en pantalla como ocurre en una animación Tween.  Para más información sobre animaciones de vistas recomendamos consultar el capítulo 4 de *El gran libro de Android* o el tutorial correspondiente en[androidcurso.com](http://androidcurso.com/index.php/269)[[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftn1). La siguiente sección repasa el uso de animaciones de vistas, pero esta vez para hacer transiciones entre actividades, y terminaremos el apartado describiendo el uso de animaciones de propiedades.  [[1]](https://poliformat.upv.es/portal/tool/a30eedda-23a5-4359-8565-dd710f2e3ea2/view_section_student#_ftnref1) <http://androidcurso.com/index.php/269> |

## 1.8.1. Animaciones de vistas: Transiciones entre actividades

|  |
| --- |
| Las animaciones de vistas puede ser una herramienta muy interesante para crear efectos en nuestras aplicaciones. Sin embargo, tienen sus inconvenientes. Hay que recordar que solo podemos usarlas para animar objetos de la clase View. Además, están limitadas a cuatro transformaciones (translación, rotación, zum y alfa) y no pueden aplicarse a otros aspectos como cambiar el color de fondo. Finalmente, solo modifica la forma en que la vista es representada, pero no sus propiedades en sí.  **Ejercicio paso a paso:** *Transiciones entre actividades.*  **1.** Crea un nuevo proyecto con nombre *TransicionActividades*.  **2.** Crea una copia de *res/layout/activity\_main.xml* en la misma carpeta con el nombre *activity\_segunda.xml*. Puedes usar las opciones del menú del botón derecho «Copiar y Pegar». Si quieres puedes añadir algún control al *layout*.  **3.** Reemplaza el código de *activity\_main.xml* por el siguiente:  <RelativeLayout 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=".MainActivity" >  <TextView  android:id="@+id/textView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_centerHorizontal="true"  android:layout\_centerVertical="true"  android:text="@string/hello\_world" />  <Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView1"  android:layout\_below="@+id/textView1"  android:onClick="sepulsa"  android:text="Lanzar actividad" />  </RelativeLayout>  Realiza una copia de *MainActivity.java* con nombre *SegundaActivity.java*.  **4.** Edita SegundaActivity.java para que el layout mostrado sea activity\_segunda.xml.  setContentView(R.layout.activity\_segunda);  **5.** Toda nueva actividad ha de ser registrada en el manifiesto. Abre *AndroidManifest.xml*, lengüeta *Application*, *Application Nodes*, botón *«Add…»* Selecciona *Activity* y pulsa *«OK»*. En la entrada *Name* pulsa *«Browse…»* y selecciona *SegundaActivity*.  **6.** Pulsa con el botón derecho en el explorador del proyecto y selecciona *New/Android XML File.* En *Resource Type* selecciona *Tween Animation* y en *File* introduce *entrada\_derecha*.  **7.** Reemplaza el contenido por el siguiente:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <translate xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:duration=*"800"*  android:fromXDelta=*"-100%"*  android:toXDelta=*"0"* />  La nueva actividad queremos que entre de izquierda a derecha. Por lo tanto, hemos programado una animación de translación, con una duración de 800 ms, partiendo de una coordenada en x de 100% hasta una coordenada de 0. Recuerda que el origen de coordenadas x está en la izquierda de la pantalla. Por lo tanto, la coordenada 100% representa situar la actividad justo a la izquierda de la pantalla y 0 ocupando toda la pantalla.  **8.** Repite el proceso anterior para crear el fichero *salida\_izquierda* con el siguiente código:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <translate xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:duration=*"800"*  android:fromXDelta=*"0"*  android:toXDelta=*"100%"* />  La actividad que abandonamos queremos que salga por la derecha. Por lo tanto, hemos programado una animación de translación, con una duración de 800 ms, partiendo de una coordenada en x de 0 hasta una coordenada de 100%. Como las dos animaciones las arrancaremos a la vez, se consigue el efecto de que la nueva actividad desplace a la otra.  **9.** Añade el siguiente método en *MainActivity.java*:  **public** **void** sepulsa(View view){  Intent i = **new** Intent(**this**, SegundaActivity.**class**);  startActivity(i);  overridePendingTransition(R.anim.*salida\_izquierda*, R.anim.*entrada\_derecha*);  }  Con el método overridePendingTransition especificamos una animación de transición explícita a realizar a continuación. Primero se indica la animación que se aplica a la vista de la actividad entrante y luego a la saliente.  **10.** Ejecuta la aplicación y comprueba su funcionamiento.  **Práctica:** *Transición de fundido.*  Partiendo del ejercicio anterior trata de crear una transición entre las actividades. De manera que la saliente se desvanezca mientras que la entrante va apareciendo.  **1.** Crea una nueva actividad *TerceraActivity.java* con un *layout* asociado a *activity\_tercera.xml*.  **2.** Crea un nuevo botón en *activity\_main.xml* que lance la nueva actividad. De momento no actives ninguna transición.  **3.** Crea un fichero de animación con nombre *entrada\_alfa.xml* con una duración de 1 segundo que modifique el valor de alfa de 0 a 1 (fromAlpha="0.0", toAlpha="1.0"). Recuerda que el valor de alfa te permite controlar el nivel de transparencia de la vista.  **4.** Crea un fichero de animación con nombre *salida\_alfa.xml* con una duración de 1 segundo que modifique el valor de alfa de 1 a 0.  **5.** Esta vez aplica el método overridePendingTransition() en la actividad entrante, inmediatamente después de setContentView(). De esta forma la transacción se produce independientemente de quien nos lance.  **Práctica:** *Transición de zoom.*  Partiendo del ejercicio anterior trata de crear una transición a una nueva actividad, de manera que la actividad saliente reduzca su tamaño hasta desaparecer en el centro de la pantalla durante medio segundo. A continuación, la actividad entrante hace la animación contraria; es decir, parte del centro de la pantalla, con un tamaño cero, hasta ocupar la totalidad de la pantalla.  A partir de la versión 4.1 (nivel API 16) hay una alternativa para definir este tipo de transiciones. Podemos indicar la animación a través de una opción de la actividad con la clase Android.App.ActivityOptions. Un objeto de este tipo puede ser añadido al *bundle* cuando arrancamos una actividad con startActivity(Intent, Bundle). Esto permite especificar una animación personalizada, incluso cuando se inicia la actividad fuera de nuestro contexto.  **Ejercicio paso a paso:** *Transiciones definidas en ActivityOptions*  **1.** Modifica el método sepulsa() en *MainActivity.java*:  **public** **void** sepulsa(View view){  Intent i = **new** Intent(**this**, SegundaActivity.**class**);  ActivityOptions opts = ActivityOptions.*makeCustomAnimation*(  **this**, R.anim.*entrada\_derecha*, R.anim.*salida\_izquierda*);  startActivity(i, opts.toBundle());  }  **2.** Verifica que el resultado es el mismo. |

### 1.8.1.1. Aplicando animaciones de vistas en Audiolibros

|  |
| --- |
| **Ejercicio paso a paso:***Aplicando animaciones de vistas en Audiolibros.*  Vamos a aplicar algo de lo aprendido al proyecto Audiolibros.  **1.** Abre el proyecto *Audiolibrosv2*.  **2.** Crea una nueva animación de vistas en el fichero *res/anim/menguar.xml* con el siguiente código:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <set xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:shareInterpolator=*"true"*>  <scale android:interpolator=*"@android:anim/decelerate\_interpolator"*  android:fromXScale=*"1.0"* android:toXScale=*"0.0"*  android:fromYScale=*"1.0"* android:toYScale=*"0.0"*  android:pivotX=*"50%"* android:pivotY=*"50%"*  android:duration=*"500"*/>  </set>  Esta animación hará que la vista reduzca su tamaño hasta desaparecer en el centro de la vista.  **3.** Abre la clase SelectorFragment y dentro del método onCreateView() reemplaza en el switch el siguiente case:  **case** 1:  Animation anim =AnimationUtils.*loadAnimation*(actividad,R.anim.*menguar*);  anim.setAnimationListener(SelectorFragment.this);  v.startAnimation(anim);  vectorLibros.remove((**int**) id);  //adaptador.notifyDataSetChanged();  **break**;  En este código creamos un objeto *Animation* desde los recursos y lo asignamos como escuchador de eventos de nuestra propia clase. No puedes usar **this** directamente ya que realmente estamos dentro de un objeto builder de la clase AlertDialog.Builder, por lo que **this** haría referencia a este objeto. En el parámetro v se nos ha pasado la vista sobre la que se ha pulsado. Asignamos la animación a esta vista y la arrancamos. Tras eliminar el libro, no queremos que el adaptador vuelva a crear las vistas. Mejor hacer este paso cuando termine la animación. Por esta razón hemos comentado la línea.  **4.** Comenta la siguiente línea y ejecuta el proyecto:  //anim.setAnimationListener(SelectorFragment.this);  Elimina un libro y observa como al terminar la animación este vuelve a aparecer. Al terminar de una animación de vistas, estas quedan como estaban al principio.  **5.** Elimina el comentario que acabas de poner.  **6.** Queremos realizar una acción cuando termine la animación; para conseguirlo implementa en la clase la interfaz AnimationListener.  **public** **class** SelectorFragment **extends** Fragment **implements** AnimationListener {  **7.** Tras introducir este cambio y pulsar *Shift-Ctrl-O*,aparecerá un error en la clase SelectorFragment ya que los métodos de esta interfaz no han sido implementados. Sitúa el cursor sobre esta clase y selecciona *Add unimplemented methods* para que Eclipse lo haga por nosotros.  **8.** Reemplaza el código del siguiente método:  @Override  **public** **void** onAnimationEnd(Animation animation) {  adaptador.notifyDataSetChanged();  }  Cuando la animación termine ya es el momento de pedirle a nuestro adaptador que recalcule las vistas, para que así desaparezca la eliminada.  **9.** Ejecuta el proyecto y verifica el resultado.  **Práctica:** *Más animaciones de vistas en Audiolibros.*  Partiendo del ejercicio anterior, trata de crear nuevas animaciones de vistas para las acciones *compartir* e *insertar* del menú contextual.  **Preguntas de repaso:** [*Animaciones de vistas*](http://www.dcomg.upv.es/~jtomas/android/test_avanzado/test1_4_Animaciones_de_vistas.html)*.* |

## 1.8.2. Animaciones de propiedades

|  |
| --- |
|  |
| A partir de la versión 3.0 de Android (nivel de API 11) se ha incorporado una nueva API para animaciones que se conoce como animación de propiedades. A diferencia de las animaciones Tween, que solo son aplicables a vistas, una animación de propiedades puede animar cualquier tipo de objetos. Además, no está restringida a las cuatro transformaciones antes vistas; sino que podemos animar cualquier propiedad del objeto. Por ejemplo, podemos hacer una animación que cambie progresivamente el color de fondo de una vista.  Otra diferencia con respecto a las animaciones de vistas es que estas últimas solo modifican la forma en que la vista es representada, pero no sus propiedades. Por ejemplo: si aplicas una animación de vistas para que un texto se desplace por la pantalla, se visualizará correctamente; pero al acabar la animación, el texto estará en el lugar inicial, lo que te obligará a implementar tu propia lógica para manejar este cambio de posición. En una animación de propiedades estará cambiando el objeto en sí, no solamente cómo se representa.  Para ver de forma rápida los tipos de animaciones que podemos hacer con esta API te recomendamos que veas el siguiente vídeo.  **Vídeo** [**Tutorial**]**:** [*Honeycomb Animation*](http://www.youtube.com/watch?v=-9nxx066eHE)***, Chet Haase.***  Los ejemplos mostrados son parte de la aplicación API Demo. En concreto, los ejemplos correspondientes al paquete com.example.android.apis.animation. Te recomendamos su estudio; pero como son algo complejos, lo mejor sería realizar primero los ejercicios de este apartado.  Para comprender esta API para animaciones hay que conocer la jerarquía de clases definida en el paquete android.animation. Se muestra en el siguiente diagrama:  C:\Users\damon\Desktop\20141403021417_s18_image002.png  ***Figura 2.*** *Jerarquía de clases para animaciones de propiedades.*  En la clase Animator se definen una serie de características básicas de toda animación, pero no suele usarse en la práctica. Vamos a estudiar con algo más de detalle el resto de estas clases. |

### 1.8.2.1. El motor básico de animación: ValueAnimator

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| La clave de cómo se realizan las animaciones de propiedades está en la clase ValueAnimator. Esta clase proporciona un sencillo motor para conseguir que, en un objeto de otra clase, cambie una de sus propiedades desde un valor inicial a otro final, a lo largo de un tiempo determinado. Como ves, se trata de algo muy genérico que no solo puedes usar para animaciones de tipo gráfico. Por ejemplo, podrías usarlo para modificar el volumen de reproducción de un objeto MediaPlayer, para que este fuera aumentando poco a poco con el tiempo. Lo primero es crear un objeto de esta clase y asignarle los valores inicial y final de la propiedad que desea animar, junto con la duración de la animación. Finalmente se llama a start() para que la animación comience. También pueden configurarse otras propiedades de la animación. A continuación, se describen las propiedades más importantes:   |  |  |  | | --- | --- | --- | | **Propiedad** | **Descripción** | **Valor por defecto** | | **values** | Lista de valores para la animación. Típicamente dos: inicial y final. | - | | **duration** | Duración de la animación. | 300 ms | | **frameDelay** | Tiempo de refresco; es decir, cada cuánto tiempo. | 10 ms | | **startDelay** | Retraso en el arranque de la animación. | 0 ms | | **interpolator** | Permite definir diferentes tipos de interpolaciones temporales, como el movimiento lineal, acelerado o decelerado. | Accelerate  Decelerate  Interpolator | | **repeatCount** | Número de repeticiones (INFINITE si nunca para). | 0 | | **repeatMode** | Cuando termina una repetición y ha de pasar a la siguiente: la reinicia (RESTART) o la repite hacia atrás (REVERSE). | RESTART |   ***Tabla 1.*** *Propiedades de* *ValueAnimator*.  **Ejercicio paso a paso:***Una sencilla animación con ValueAnimator.*  En este ejemplo vamos a realizar una animación sobre la propiedad TextSize de un TextView.  **1.** Crea un nuevo proyecto con los siguientes datos:  Application Name: ValueAnimator  Minimum Required SDK: API 11 (3.0)  **2.** Edita el *layout* *activity\_main* y añade el atributo id al TextView:  <TextView  android:id=*"@+id/text\_view"*  ... />  **3.** Edita el código de la actividad y añade el siguiente código:  public class MainActivity extends Activity  implements ValueAnimator.AnimatorUpdateListener {  private TextView textView;    @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  textView = (TextView) findViewById(R.id.text\_view);  ValueAnimator animacion = ValueAnimator.ofFloat(10, 40);  animacion.setDuration(1000);  animacion.setInterpolator(new DecelerateInterpolator());  animacion.setRepeatCount(4);  animacion.setRepeatMode(ValueAnimator.REVERSE);  animacion.addUpdateListener(this);  animacion.start();  }    @Override  public void onAnimationUpdate(ValueAnimator animacion) {  float value =((Float) (animacion.getAnimatedValue())).floatValue();  textView.setTextSize(value);  }  }  En el método onCreate() creamos el objeto animacion indicando el valor inicial y final a animar. En lugar de dos valores también se podría indicar una lista de valores para que la animación pase por todos los valores. A continuación, se configuran otros parámetros de la animación. Puedes consultar la tabla anterior para ver su significado. Hay que advertir que con un valor de repeticiones igual a 4, la animación se realizará 5 veces. En la penúltima línea indicamos que nuestra clase recogerá los eventos generados por la animación, y en la última arrancamos la animación.  El siguiente método corresponde a la interfaz AnimatorUpdateListener. Será llamado cada vez que el objeto animacion quiera cambiar el valor a animar. El parámetro que se recibe es la animación que genera el evento. En este método nos limitamos a recoger el valor actual de la animación con el método getAnimatedValue() y a asignarlo a la propiedad TextSize de nuestra vista.  **4.** Ejecuta el proyecto para verificar el resultado.  Además del utilizado en el ejemplo, también podemos registrar cuatro escuchadores de eventos de la interfaz Animator.AnimatorListener para controlar cuándo comienza, si se produce alguna repetición, si termina o si se cancela la animación. Puedes usar esta interfaz en los descendientes de Animator, entre los que se incluyen las clases que veremos a continuación:     |  |  | | --- | --- | | **Interfaz** | **Métodos** | | Animator.AnimatorListener | onAnimationStart()  onAnimationRepeat()  onAnimationEnd()  onAnimationCancel() | | ValueAnimator.AnimatorUpdateListener | onAnimationUpdate() |   ***Tabla 2.*** *Escuchadores de eventos en animación de propiedades.* |

### 1.8.2.2. Automatizando las animaciones: ObjectAnimator

|  |
| --- |
|  |
| El uso de ValueAnimator puede ser bastante engorroso. Se limita a generar un evento cada vez que ha de cambiar un valor y nosotros hemos de cambiar este valor a mano. La finalidad de ObjectAnimator es automatizar esta tarea.  Se trata una subclase de ValueAnimator que le permite establecer un objeto de destino y la propiedad que deseas animar. Esta clase actualiza automáticamente la propiedad cada vez que se calcula un nuevo valor. Por ejemplo, para animar una propiedad de un objeto desde el valor 10 al 40 escribiremos:  ObjectAnimator animacion = ObjectAnimator.*ofFloat*(objeto, "propiedad", 10, 40);  Y no nos tendremos que preocupar de capturar ningún evento.  **Ejercicio paso a paso:***Una sencilla animación con ObjectAnimator.*  En este ejemplo vamos a realizar la misma animación que hicimos con ValueAnimator. Como verás resulta más sencillo con esta clase.  **1.** Crea una copia del proyecto anterior (selecciónalo, pulsa «Ctrl+C» y «Ctrl+V»). Indica el nombre *ObjectAnimator*.  **2.** En la actividad, reemplaza la línea:  ValueAnimator animacion = ValueAnimator.*ofFloat*(10,40);  Por:  ObjectAnimator animacion = ObjectAnimator.*ofFloat*(textView, "textSize", 10, 40);  **3.** Ya no necesitamos recoger eventos, en la declaración de la clase elimina:  **implements** ValueAnimator.AnimatorUpdateListener  **4.** Aparecerán una serie de errores. Has de eliminar todo el método onAnimationUpdate().  **5.** Elimina también el registro del escuchador de eventos dentro de onCreate():  animacion.addUpdateListener(**this**).  **6.** Verifica que el resultado es idéntico al ejercicio anterior.  Si ObjectAnimator es más sencilla de usar, ¿por qué explicar ValueAnimator? ObjectAnimator tiene algunas restricciones, como la exigencia de que el objeto de destino tenga algunos métodos específicos. Por lo tanto, en algunas ocasiones puede ser más práctico utilizar ValueAnimator directamente. Las restricciones que nos impone ObjectAnimator para animar un objeto son cuatro:  a) La propiedad que se está animando debe tener una función *setter* en forma de set<Propiedad>(). En el ejemplo anterior la propiedad es "textSize"por lo que el objeto ha de tener el método setTextSize(). Si este método *setter* no existe, tienes tres alternativas:   Agrega el método *setter* a la clase, siempre que tengas derechos.   Utiliza una clase contenedora (*wrapper*) para recibir el valor con un método *setter* válido y modificar el objeto original.   Utiliza ValueAnimator.  b) Si en la construcción del ObjectAnimator, en la lista de valores que ha de tomar la propiedad, solo se indica uno, la animación irá desde el valor actual de la propiedad hasta el indicado. Por lo tanto, esta propiedad debe de poder ser consultada a través de una función *getter*. Ha de tener la forma get<Propiedad>().  c) Los métodos *setter* y *getter* han de trabajar con el mismo tipo especificado en la creación del ObjectAnimator. Disponemos de métodos ofFloat() y ofInt() para trabajar con tipos simples, ofObject() cuando los valores son objetos y [ofPropertyValuesHolder](http://developer.android.com/reference/android/animation/ObjectAnimator.html#ofPropertyValuesHolder(java.lang.Object, android.animation.PropertyValuesHolder...))() cuando queremos trabajar con varias propiedades diferentes en paralelo.  d) Tenemos que asegurarnos que cada vez que se cambia una propiedad se llama al método invalidate() para forzar el redibujado. Los métodos *setters* de las vistas llaman a invalidate(). Sin embargo, otros tipos de objeto como los Drawable no siempre lo hacen. En estos casos crea un escuchador de evento para onAnimationUpdate() y redibuja manualmente el objeto. |

### 1.8.2.3. Combinando animaciones: AnimatorSet

|  |
| --- |
|  |
| La clase AnimatorSet proporciona un mecanismo para reproducir un grupo de animaciones conjuntamente, de forma que se ejecuten unas en relación a las otras. Puede ejecutar animaciones en paralelo, de forma secuencial o después de un tiempo de espera especificado. Imaginemos que queremos secuenciar cuatro animaciones de la siguiente manera: primero la animación 1, luego la 2 y la 3 simultáneamente y cuando termine la 3, que arranque la 4. El código que tendríamos que escribir es el siguiente:  AnimatorSet conjunto = **new** AnimatorSet();  conjunto.play(anim1).before(anim2);  conjunto.play(anim2).with(anim3);  conjunto.play(anim4).after(anim3);  conjunto.start();  Donde anim1,…, anim4 son objetos de las clases ValueAnimator, ObjectAnimator o incluso AnimatorSet (en realidad cualquier descendiente de Animator). |

### 1.8.2.4. Definiendo animaciones en XML

|  |
| --- |
|  |

|  |
| --- |
|  |
| La definición de animaciones en una aplicación suele ser responsabilidad de los diseñadores gráficos, no de los programadores. Por esta razón, el sistema de animación de propiedades permite declarar las animaciones desde ficheros XML.  Para diferenciar estos ficheros de los utilizados en las animaciones de vistas, desde la versión 3.1 de Android, has de usar la carpeta *res/animator* en lugar de*res/anim*. Se trata solo de una convención; podrás comprobar que en muchas ocasiones se utiliza la carpeta *res/anim* para almacenar animaciones de propiedades.  Podemos definir objetos de las tres clases estudiadas definiendo ficheros XML cuya etiqueta raíz sea una de las que se muestran en la siguiente tabla:     |  |  | | --- | --- | | **Etiqueta** | **Clase** | | **<animator>** | ValueAnimator | | **<objectAnimator>** | ObjectAnimator | | **<set>** | AnimatorSet |   ***Tabla 3.*** *Etiquetas XML en animación de propiedades.*  La sintaxis que podemos usar en <animator> se muestra a continuación:  <animator  android:duration="*int*"  android:valueFrom="*float* | *int* | *color*"  android:valueTo="*float* | *int* | *color*"  android:startOffset="*int*"  android:interpolator="tipo de interpolación"  android:repeatCount="*int*"  android:repeatMode=["repeat" | "reverse"]  android:valueType=["intType" | "floatType"]/>  Para <objectAnimator> tendría los mismos atributos que <animator> más:  <objectAnimator  android:propertyName="*string*"  … />  Para <objectAnimator> usaríamos la siguiente sintaxis:  <set android:ordering=["together" | "sequentially"]  android:interpolator="tipo de interpolación">   <animator … />   <objectAnimator … />   <set>  …  </set>  </set>  La siguiente tabla muestra los tipos de interpolación que podemos indicar en el atributo interpolator:   |  | | --- | | **Recurso** | | **@android:anim/accelerate\_decelerate\_interpolator** | | **@android:anim/accelerate\_interpolator** | | **@android:anim/anticipate\_interpolator** | | **@android:anim/anticipate\_overshoot\_interpolator** | | **@android:anim/bounce\_interpolator** | | **@android:anim/cycle\_interpolator** | | **@android:anim/decelerate\_interpolator** | | **@android:anim/linear\_interpolator** | | **@android:anim/overshoot\_interpolator** |   ***Tabla 4.*** *Valores que podemos asignar al atributo* interpolator.  **Ejercicio paso a paso:** *Definir una animación con XML.*  **1.** Crea un nuevo proyecto con los siguientes datos:  Application Name: AnimacionXML  Minimum Required SDK: API 11 (3.0)  **2.** Pulsa con el botón derecho sobre el proyecto y selecciona *New/Android XML File*. En *Resource Type* indica *Property Animation* y en *File* introduce *animacion.xml*. Pulsa el botón «*Finish»*. Se creará el fichero *res/animator/animacion.xml*. Reemplaza su contenido por:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <set xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:ordering=*"sequentially"* >  <set>  <objectAnimator  android:duration=*"1000"*  android:propertyName=*"textSize"*  android:valueTo=*"50"*  android:valueType=*"floatType"* />  <objectAnimator  android:duration=*"2000"*  android:propertyName=*"textScaleX"*  android:valueFrom=*"0.2"*  android:valueTo=*"1.5"* />  </set>  <objectAnimator  android:duration=*"2000"*  android:propertyName=*"textColor"*  android:repeatCount=*"infinite"*  android:repeatMode=*"reverse"*  android:valueFrom=*"#000000"*  android:valueTo=*"#0000FF"* />  </set>  **3.** Edita el *layout* *activity\_main* y añade el atributo id al TextView:  <TextView  android:id=*"@+id/text\_view"*  ... />  **4.** Añade en la actividad al final del método onCreate():  TextView textView = (TextView) findViewById(R.id.*text\_view*);  AnimatorSet set = (AnimatorSet) AnimatorInflater.*loadAnimator*(**this**, R.animator.*animacion*);  set.setTarget(textView);  set.start();  **5.** Comprueba el resultado. |

### 1.8.2.5. Nuevas propiedades de la clase View

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Como acabamos de comprobar, las animaciones de propiedades nos permiten animar cualquier propiedad de un objeto, no solo las cuatro transformaciones que permitían las animaciones de vistas. Sin embargo, es posible que quieras desplazar o rotar la vista anterior y no sepas cómo hacerlo, dado que un TextView no dispone de propiedades para esto. Para superar este obstáculo, en la versión 3.0 se han añadido una serie de propiedades a la clase View que podemos modificar dinámicamente. Se muestran en la siguiente tabla:   |  |  | | --- | --- | | **Propiedad** | **Descripción** | | **Alpha** | Nivel de transparencia (0=transparente, 1=opaca) | | **translationX** | Desplazamiento X respecto a su posición en *layout* | | **translationY** | Desplazamiento Y respecto a su posición en *layout* | | **Rotation** | Rotación en grados de la vista (siguiendo agujas reloj) | | **scaleX** | Factor de ampliación de la vista en el eje X | | **scaleY** | Factor de ampliación de la vista en el eje Y | | **pivotX** | Posición X de pivote para rotaciones y escalados | | **pivotY** | Posición Y de pivote para rotaciones y escalados | | **rotationX** | Rotación 3D en grados de la vista sobre el eje X | | **RotationY** | Rotación 3D en grados de la vista sobre el eje Y | | **cameraDistance** | Distancia en eje Z para calcular la perspectiva 3D | | **X** | Posición desde la izquierda del padre | | **Y** | Posición desde arriba del padre |   ***Tabla 5.*** *Nuevas propiedades de* View *añadidas en la versión 3.0.*  Puedes utilizar el método *setter* (set+Propiedad) y *getter* (get+Propiedad) para acceder a estas propiedades. Todas son de tipo float.  **Ejercicio paso a paso:** *Uso de propiedades para animaciones con vistas.*  **1.** Abre el proyecto anterior.  **2.** Crea un nuevo fichero de animación de propiedades con el nombre *res/animator/animacion2.xml*. Reemplaza su contenido por:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <set xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:ordering=*"sequentially"*>  <set>  <objectAnimator  android:propertyName=*"x"*  android:duration=*"800"*  android:valueTo=*"500"*  android:valueType=*"floatType"*/>  <objectAnimator  android:propertyName=*"y"*  android:duration=*"800"*  android:valueTo=*"400"*  android:valueType=*"floatType"*/>  </set>  <objectAnimator  android:propertyName=*"alpha"*  android:duration=*"800"*  android:repeatCount=*"infinite"*  android:repeatMode=*"reverse"*  android:valueFrom=*"1"*  android:valueTo=*"0.2"*/>  </set>  **3.** Asigna esta animación al TextView y verifica su funcionamiento.  Si te fijas en la tabla de nuevas propiedades añadidas en la versión 3.0, algunas nos permiten hacer efectos 3D. Para orientarte has de saber que el eje X es la dimensión en altura y el eje Y es la dimensión en anchura. Para el eje Z imagina que estás mirando la vista desde arriba, la distancia desde tus ojos a la vista sería la dimensión Z. Esta distancia puede ser controlada con la propiedad cameraDistance. Valores pequeños aumentan la sensación de perspectiva. Puedes girar la vista con rotationX. Si estuvieras muy cerca, este giro te daría la impresión de que la vista te va a golpear en la frente o barbilla. Si la giraras con rotationY, te daría la impresión de que te va a golpear en una oreja. Con pivotX y pivotY puedes situar el centro de giro.  **Práctica:** *Uso de propiedades para animaciones 3D.*  En esta práctica vamos a modificar el ejercicio anterior para realizar una animación en 3D. Lo primero es ampliar el tamaño del texto para que la animación sea más atractiva. Imagina que el texto está colgado de una cuerda de tender imaginaria por su parte superior y la parte inferior se desplaza hacia delante y hacia atrás de forma indefinida. Crea un nuevo fichero de animación con el nombre *animacion3D.xml* que realice este efecto.  NOTA: Si queremos que el texto gire usando como eje la parte superior, tendríamos que conseguir que pivotX fuera 0. Sin embargo, en la etiqueta <objectAnimator> no existe ningún atributo para cambiar este pivote. Entonces, cómo podemos cambiar el valor de pivotY. La solución puede ser introducir una nueva animación para cambiar este valor, tal y como se muestra a continuación:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <set xmlns:android=*"http://schemas.android.com/apk/res/android"*>  <objectAnimator  android:propertyName=*"pivotY"*  android:valueTo=*"0"*/>  <objectAnimator  android:propertyName=*"rotationX"*  android:duration=*"2000"*  …/>  </set>  También sería interesante poner el valor de pivotX en el 50 % del ancho del texto. A pesar de que en las animaciones de vistas podíamos usar porcentajes (%), en las animaciones de propiedades no está permitido. Puedes tratar de situar el centro usando medidas como dp o px. |

### 1.8.2.6. Aplicando animaciones de propiedades en Audiolibros

|  |
| --- |
| **Ejercicio paso a paso:***Aplicando animaciones de propiedades en Audiolibros.*  En una sección anterior aplicamos animaciones de vistas en el proyecto Audiolibros. Ahora vamos a aplicarle animaciones de propiedades. El proceso es muy similar; pero ahora tendremos algunas ventajas: como poder animar cualquier propiedad de cualquier objeto o aplicar los nuevos efectos 3D.  **1.** Abre el proyecto *Audiolibrosv3*.  **2.** Crea una nueva animación de vistas en el fichero *res/animator/menguar.xml* con el siguiente código:  <?xml version=*"1.0"* encoding=*"utf-8"*?>  <set xmlns:android=*"http://schemas.android.com/apk/res/android"* >  <objectAnimator  android:propertyName=*"scaleX"*  android:duration=*"500"*  android:valueTo=*"0"*  android:valueType=*"floatType"*/>  <objectAnimator  android:propertyName=*"scaleY"*  android:duration=*"500"*  android:valueTo=*"0"*  android:valueType=*"floatType"*/>  </set>  El resultado de esta animación será idéntico al definido en *res/anim/menguar.xml*.  **3.** Abre la clase SelectorFragment y dentro del método onCreateView() reemplaza en el switch el siguiente case:  case 1:  Animator anim = AnimatorInflater.*loadAnimator*(actividad, R.animator.*menguar*);  anim.addListener(SelectorFragment.this);  anim.setTarget(v);  anim.start();  vectorLibros.remove((**int**) id);  //adaptador.notifyDataSetChanged();  break;  En este código es muy similar al realizado con animaciones de vistas. Creamos un objeto *Animator* desde los recursos y le asignamos como escuchador de eventos nuestra propia clase. Asignamos la animación a esta vista y la arrancamos. En este caso no es necesario forzar el repintado de la vista con invalidate().  **4.** Comenta la siguiente línea y ejecuta el proyecto:  //anim.addListener(SelectorFragment.this);  Elimina un libro y observa como al terminar la animación, el libro ya no está. Al terminar una animación de propiedades todos los objetos quedan con su valor final.  **5.** Elimina el comentario que acabas de poner.  **6.** Igual que antes, queremos actualizar el adaptador cuando termine la animación. Para conseguirlo implementa en la clase la interfaz AnimatorListener:  **public** **class** SelectorFragment **extends** Fragment **implements** AnimatorListener {  **7.** Tras introducir este cambio y pulsar *Shift-Ctrl-O*, aparecerá un error en la clase SelectorFragment ya que los métodos de esta interfaz no han sido implementados. Sitúa el cursor sobre esta clase y selecciona *Add unimplemented methods* para que Eclipse lo haga por nosotros.  **8.** Si has realizado el ejercicio de animaciones de vistas, elimina los tres métodos de esta interfaz que ya no implementamos. Los identificarás rápidamente al estar marcados como errores.  **9.** Reemplaza el código del siguiente método:  @Override **public** **void** onAnimationEnd(Animator animation) {  adaptador.notifyDataSetChanged();  adaptador.recalculaFiltro();  }  **10.** Ejecuta el proyecto y verifica que tras borrar un elemento el resultado no es satisfactorio. La vista que ocupa el lugar de la que acabamos de borrar ha desaparecido. La razón está en que al definir el adaptador del GridView estamos reciclando las vistas que ya estaban creadas con anterioridad (parámetro convertView).  **11.** Abre la clase AdaptadorLibros y estudia cómo se crean las vistas en el método getView(). La primera vez que se cree el GridView el parámetro convertView será **null**, por lo que la vista es creada desde XML. Las siguientes veces ya tendremos una vista creada en vistaReciclada, por lo que la reciclaremos para ganar tiempo. El problema está en que si la vista que reciclamos tiene un factor de escala 0, la nueva vista también lo tendrá.  **12.** Para resolver este problema añade la sección else que se muestra debajo:  **if** (vistaReciclada == **null**) {  vistaReciclada = inflador.inflate(R.layout.*elemento\_selector*, **null**);  } **else {**  **vistaReciclada.setScaleX(1);**  **vistaReciclada.setScaleY(1);**  **}**  **13.** Ejecuta el proyecto y verifica que ahora funciona el borrado de libros.  **Práctica:** *Más animaciones de propiedades en Audiolibros.*  Partiendo del ejercicio anterior, trata de crear nuevas animaciones de propiedades para las acciones *compartir* e *insertar* del menú contextual.  **Preguntas de repaso:**[*Animaciones de propiedades*](http://www.dcomg.upv.es/~jtomas/android/test_avanzado/test1_5_Animaciones_de_propiedades.html)*.*  **Test de evaluación:**  Una vez domines los conceptos desarrollados en esta unidad puedes realizar el test de evaluación. Para ello accede al menú Exámenes, en poli[Format]. El examen estará disponible hasta el jueves de la semana que viene. Te recomendamos que realices el examen antes de continuar con la siguiente unidad.  ***Nota:*** *Los exámenes de evaluación se activan tras la clase de repaso de esta unidad.* |