Anatomia y Fundamentos de las Aplicaciones

Curso de Desarrollo en Android

GSyC/LibreSoft

Marzo de 2012

Contenidos

Contenidos

Componentes de las Aplicaciones Android

- Actividades *(Activities)*: Cada pantalla de una aplicación. Utilizan Vistas *(Views)* como componentes que muestran información y responden a las acciones del usuario
- Servicios *(Services)*: Componentes de la aplicación que se ejecutan de forma invisible, actualizando los datos y las Actividades, y disparando Notificaciones. Realizan el procesamiento normal de la aplicación que debe continuar incluso cuando las Actividades de la aplicación no están visibles.
- Proveedores de Contenidos *(Content Providers)*: Almacenes de datos compartidos. Gestionan las Bases de Datos para las aplicaciones.
- Intenciones *(Intents)*: Mecanismo que permite el paso de mensajes destinados a ciertas Actividades o Servicios, o a todo el sistema (Intenciones de *broadcast*). Exponen la intención de que se haga algo. El sistema determinará el destinatario que lo efectuará.
- Receptores de *Broadcast (Broadcast Receivers)*: Los crean las aplicaciones como consumidores de las Intenciones de *broadcast* que cumplan ciertos criterios.
- Notificaciones *(Notifications)*: Mecanismo que permite a las aplicaciones señalar algo a los usuarios sin interrumpir la Actividad en primer plano.

El Manifiesto de una Aplicación (I)

- Cada Aplicación incluye un fichero de Manifiesto, `AndroidManifest.xml`.
- Define la estructura de la aplicación y sus componentes.
- Incluye un nodo raíz y un nodo para cada uno de sus tipos de componentes.
- A través de **filtros de intenciones** y **permisos** determina cómo interactuará con otras aplicaciones.
- Nodo raíz `manifest`: incluye el nombre del paquete de la aplicación:

```
1  <manifest xmlns:android=http://schemas.android.com/apk/res/android
           package="com.my_domain.my_app">
3          [ ... manifest nodes ... ]
   </manifest>
```

El Manifiesto de una Aplicación (II)

- Nodo `application`:
  - Indica metadatos de la aplicación (título, icono, tema)
  - Contiene los nodos de actividades, servicios, proveedores de contenidos y receptores de broadcast.

```
<application android:icon="@drawable/icon"
             android:theme="@style/my_theme">
  <activity android:name=".MyActivity" android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>

  <activity ... >
    ...
  </activity>

  <service ... >
    ...
  </service>

  ...

</application>
```

El Manifiesto de una Aplicación (III)

- Nodo `uses-permission`:
  - Declara los permisos que la aplicación necesita para operar.
  - Serán presentados al usuario durante la instalación para que los acepte o deniegue.
  - Se requieren permisos para utilizar muchos de los servicios nativos Android (enviar SMS, hacer llamadas, usar servicios de localización...)
  - Permisos más habituales: `INTERNET`, `READ_CONTACTS`, `WRITE_CONTACTS`, `RECEIVE_SMS`, `SEND_SMS`, `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
</uses-permission>
```

El Manifiesto de una Aplicación (IV)

- Nodo `permission`:
  - Define un permiso que se requiere para que otras aplicaciones puedan acceder a partes restringidas de la aplicación
  - Las otras aplicaciones necesitarán poner un `uses-permission` en su Manifiesto para utilizar este permiso

```
<permission android:name="com.clau.DETONATE_DEVICE"
2           android:protectionLevel="dangerous"
            android:label="Self_Destruct"
4           android:description="@string/detonate_description">
</permission>
```

El Manifiesto de una Aplicación (V)

- Nodo `instrumentation`:
  - Permite definir tests de ejecución para las Actividades y Servicios.

```
<instrumentation android:label="My Test"
                 android:name=".MyTestClass"
                 android:targetPackage="com.clau.aPackage">
</instrumentation>
```

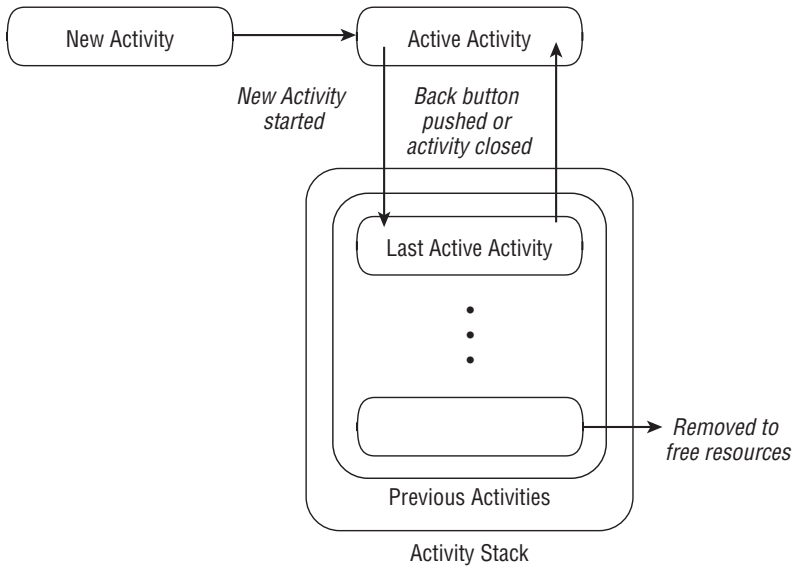
- Más información sobre el manifiesto en:

  `http://code.google.com/android/devel/bblocks-manifest.html`

Contenidos

Aplicaciones, Actividades y Pila de Actividades

- Las Aplicaciones Android no son iguales a las aplicaciones en los sistemas operativos tradicionales: sólo hay una en primer plano que normalmente ocupa toda la pantalla.
- Las Aplicaciones están formadas por Actividades (pantallas).
- Al arrancar una nueva aplicación, pasa a primer plano situando una Actividad encima de la que hubiera, formándose así una Pila de Actividades.
- El botón *Back* (↩) cierra la Actividad en primer plano y recupera la de la cima de la Pila (cerrando la aplicación en su caso).



Aplicaciones y procesos

- Las aplicaciones Android no tienen control sobre su propio ciclo de vida:
  - deben estar pendientes de posibles cambios en su estado y reaccionar como corresponda
  - en particular, deben estar preparadas para su terminación en cualquier momento
- En general, cada aplicación Android se ejecuta en un proceso que ejecuta una instancia de Dalvik.
- Android mata sin previo aviso los procesos que considera necesarios cuando lo considera necesario para mantener el sistema responsivo.
- El *runtime* de Android gestiona el proceso de cada aplicación, y por extensión de cada Actividad que contenga.

Aplicaciones, Actividades, Vistas, Diseños

- Una Actividad representa una pantalla que una aplicación muestra a sus usuarios (equivalente al concepto de *Form* en aplicaciones de escritorio convencionales).
- La mayoría de las Actividades ocupan toda la pantalla del dispositivo, pero pueden crearse actividades semi-transparentes, flotantes o que utilizan cajas de diálogo.
- Para crear una nueva Actividad en una aplicación se hereda de la clase `Activity`.
- Las Interfaz de Usuario de una Actividad se crea mediante Vistas *(Views)* (equivalentes al concepto de *Widgets* en aplicaciones de escritorio convencionales).
- Las Vistas se agrupan en Diseños *(Layouts)* que mostrará la aplicación.

Creación de una Actividad

src/com.clau.myapplication/MyActivity.java

```
1  package com.clau.myapplication;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class MyActivity extends Activity {
7    /** Called when the activity is first created. */
8    @Override
9    public void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.main);
12   }
13 }
```

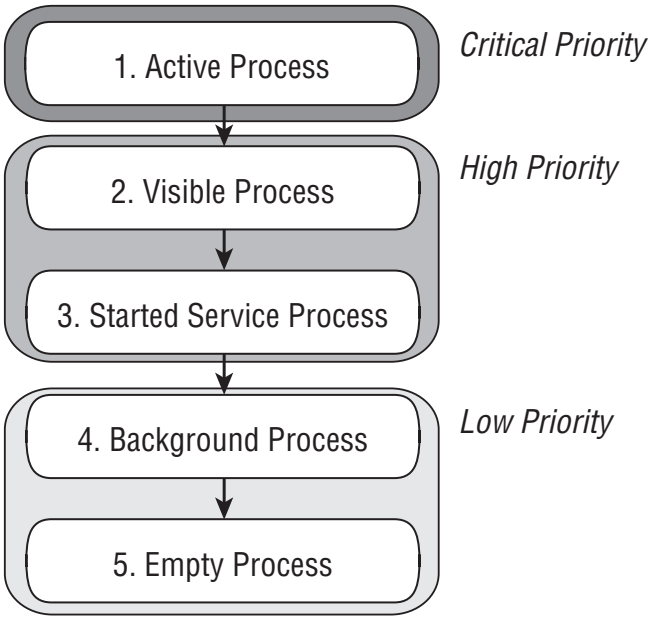
- Para usar una Actividad en una Aplicación, hay que registrarla en el Manifiesto, incluyendo las Intenciones a las que responderá:

AndroidManifest.xml

```
1  <activity android:label="@string/app_name"
2            android:name=".MyActivity">
3    <intent-filter>
4      <action android:name="android.intent.action.MAIN" />
5      <category android:name="android.intent.category.LAUNCHER" />
6    </intent-filter>
7  </activity>
```

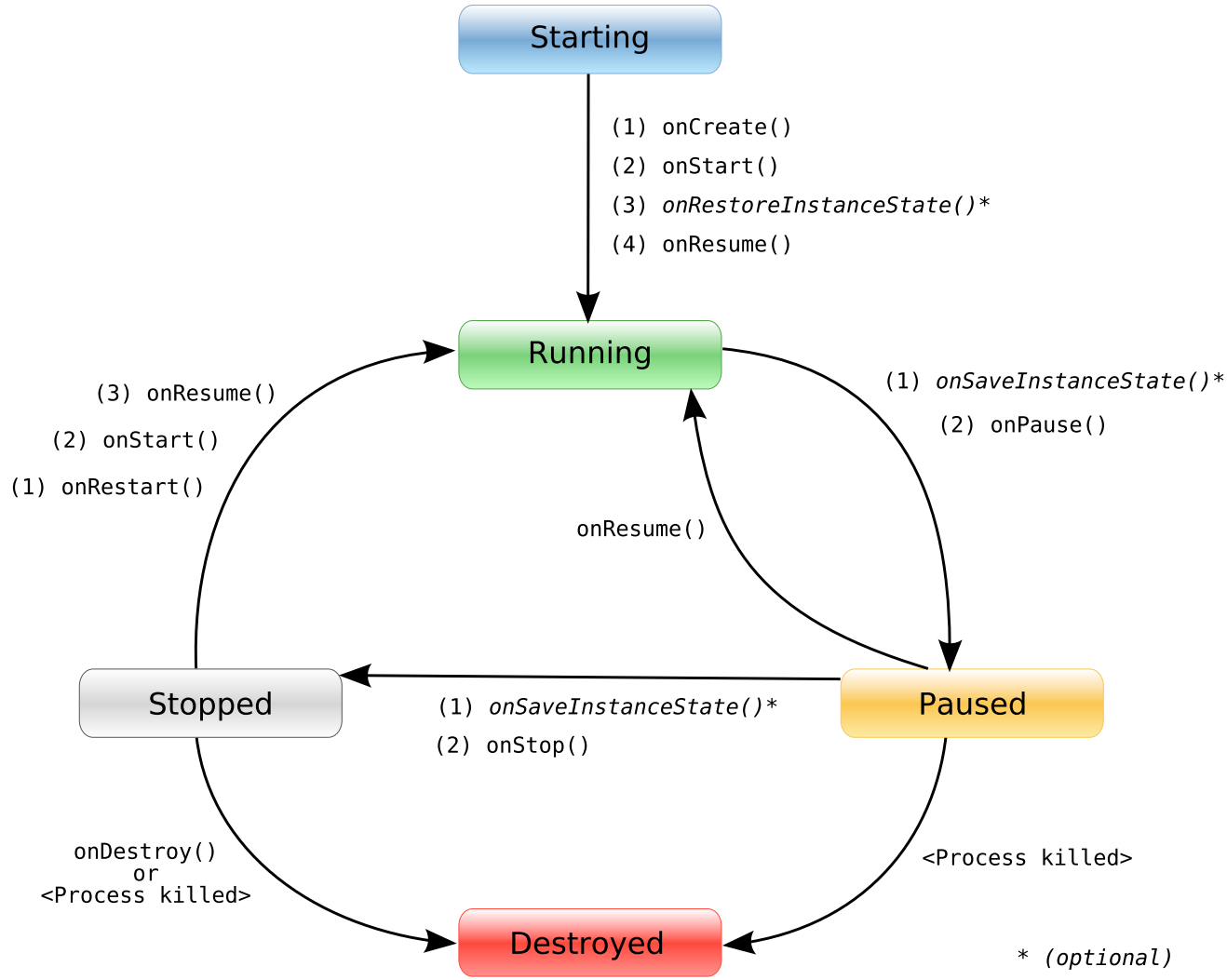
Procesos y prioridades

- El orden en que los procesos se van matando para liberar recursos lo determina las prioridades de la aplicaciones que alojan.
- Árbol de prioridades:



- Prioridad Crítica:
  - **Procesos activos**: Los que están interactuando con el usuario). Hay muy pocos y se les mata sólo como último recurso.
- Prioridad Alta:
  - **Procesos visibles**: Visibles, pero no activos (p.ej, parcialmente tapados por otra actividad). Hay muy pocos y se les mata sólo en circunstancias extremas.
  - **Procesos de servicios arrancados**: Procesamiento que debe continuar aunque no tenga interfaz visible. Se les mata sólo en circunstancias extremas.
- Prioridad Baja:
  - **Procesos de fondo**: Alojan actividades que no caen en las categorías anteriores. Hay muchos. Se les mata cuando se necesitan recursos para el resto de procesos.
  - **Procesos vacíos**: Android mantiene en memoria las aplicaciones que han terminado, por si son relanzadas. Se les mata rutinariamente.

Estados de una Actividad (I)



Estados de una Actividad (II)

- Activo *(Running)*: La Actividad está encima de la pila, es visible, tiene el foco (recibe la entrada del usuario). Cuando otra Actividad pase a estar activa, ésta pasará a estar pausada.
- Pausado *(Paused)*: La Actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra Actividad transparente o que no ocupa toda la pantalla. Cuando una Actividad es tapada por completo pasa a estar parada.
- Parado *(Stopped)*: Cuando la Actividad no es visible. Permanece en memoria reteniendo su estado. Cuando una actividad entra en parada puede ser bueno que salve todos sus datos y el estado de la Interfaz de usuario.
- Destruido *(Destroyed)*: Cuando la Actividad termina, o es matada por el *runtime* de Android. Sale de la Pila de Actividades. Necesita ser reiniciada para volver a estar activa.

Estados de una Actividad (III)

- En la clase `Activity` existen métodos para ser redefinidos *(overrided)* en sus clases derivadas para incluir el código a ejecutar en las transiciones entre estados: `OnCreate, onStart, onPause, onStop...`
- Los métodos redefinidos siempre deben llamar al método de la superclase:

```
1  // Called at the start of the visible lifetime.
   @Override
3  public void onStart(){
     super.onStart();
5    // Apply any required UI change now that the Activity is visible.
   }
```

Métodos de transición entre estados (I)

- `onCreate(Bundle)`:
  - Se invoca cuando la Actividad se arranca por primera vez.
  - Se utiliza para tareas de inicialización a realizar una sola vez, como crear la interfaz de usuario de la Actividad.
  - Su parámetro es `null` o información de estado guardada previamente por `onSaveInstanceState()`.
- `onStart()`:
  - Se invoca cuando la Actividad va a ser mostrada al usuario.
- `onResume()`:
  - Se invoca cuando la Actividad va a empezar a interactuar con el usuario.

Métodos de transición entre estados (II)

- `onPause()`:
  - Se invoca cuando la actividad va a pasar al fondo porque otra actividad ha sido lanzada para ponerse delante.
  - Se utiliza para guardar el estado persistente de la Actividad
- `onStop()`:
  - Se invoca cuando la actividad va a dejar de ser visible y no se necesitará durante un tiempo.
  - Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la Actividad ser destruida directamente.
- `onRestart()`:
  - Se invoca cuando la Actividad va a salir del estado de parada para volver a estar activa.
- `onDestroy()`:
  - Se invoca cuando la Actividad va a ser destruida.
  - Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la Actividad ser destruida directamente.

Métodos de transición entre estados (III)

- `onSaveInstanceState(Bundle)`:
  - Se invoca para permitir a la actividad guardar su estado, p.ej: la posición del cursor en una caja de texto.
  - Normalmente no necesita ser redefinido porque la implementación de la clase `Activity` ya guarda todo el estado de todos los componentes de la Interfaz de Usuario.
- `onRestoreInstanceState(Bundle)`:
  - Se invoca para recuperar el estado guardado por `onSaveInstanceState()`.
  - Normalmente no necesita ser redefinido porque la implementación de la clase `Activity` ya recupera todo el estado de todos los componentes de la Interfaz de Usuario.

Contenidos

Tipos de recursos

- Hay distintos tipos de recursos, que se definen en ficheros XML alojados en una cierta subcarpeta de `res`:
  - Valores simples (carpeta `values`): Strings, colores y dimensiones.
  - Recursos dibujables (carpeta `drawable`): ficheros con imágenes, incluyendo el icono de la aplicación
  - Animaciones (carpeta `anim`)
  - Menús (carpeta `menu`)
  - Diseños (carpeta `layout`)
  - Estilos (carpeta `values`)
  - Varios (carpeta `xml`)

Valores simples

- Definen Strings, colores o dimensiones
- Se definen como un elemento:

```
<color name="opaque_red">#f00</color>
```

- Las dimensiones pueden tener distintas unidades:
  `px, in, mm, pt, dp, sp`
  - `dp`: *Density-independent pixels* (también `dip`), pixels independientes de los dpi (puntos por pulgada) del dispositivo. Son un valor relativo a un dispositivo de 160dpi.
  - `sp`: *Scale-independent pixels*, igual que los `dp`, pero también escalados al tamaño de *font* elegido por el usuario.

```
1 <dimen name="sixteen_sp">16sp</dimen>
```

- Un fichero XML contiene la definición de uno o más de estos elementos.
- Todos estos recursos se identifican con el valor de su atributo `name`.

Drawables

- Para ficheros de *bitmaps* o de imágenes "estirables" (`.9.png`)
- También puede definirse uno de estos recursos como un recuadro de color:

```
1 <drawable name="solid_blue">#0000ff</drawable>
```

(un fichero XML puede contener uno o más recuadros de color)

- Los ficheros se identifican con su nombre de fichero, y los recuadros de color con el valor de su atributo `name`.

Animaciones

- Para realizar animaciones sencillas sobre uno o varios gráficos: rotaciones, *fading*, movimiento y estiramiento.
- Cada animación se define en un fichero XML
- Estos recursos se identifican con su nombre de fichero.

Menús

- Existen tres tipos de menús: Menú de opciones, menú contextual y submenú
- Un menú de opciones o contextual se define en un fichero XML.
- Un submenú se define dentro de otro menú.
- Un menú de opciones o contextual se identifica con su nombre de fichero.
- Un submenú se identifica con el valor de su atributo `id`.
- Un elemento de un menú también puede ser identificado con el valor de su atributo `id`.

Layouts

- Un *layout* se define en un fichero XML.
- Dentro de un *layout* se definen los elementos que lo componen: *Views*, *ViewGroups*
- Un *layout* se identifica con su nombre de fichero.
- Un elemento de un *layout* también puede ser identificado con el valor de su atributo `id`.

Estilos

- Un estilo es uno o más atributos que se aplican a un elemento.
- Un tema es uno o más atributos que se aplican a todo lo que hay en pantalla. Un tema se asigna como atributo a una Actividad en su Manifiesto.
- Se definen dentro en un elemento `<style>` que contiene strings, colores, o referencias a otros recursos.
- Un estilo entero se referencia con el valor de su atributo `name`.
- Un elemento dentro de un estilo también puede ser referenciado de la manera adecuada según el tipo de recurso.

Recursos para diferentes idiomas y diferente hardware

- Pueden especificarse recursos alternativos para diferentes idiomas o diferente hardware con subcarpetas con el mismo nombre que la original más un sufijo o conjunto de sufijos.
- Ejemplos:
  - Valores simples para idioma español: `values-es`
  - *Layouts* para terminal girado: `layout-land`
  - *Layouts* para teclado sobreimpresionado: `layout-keysexposed`
- Pueden ponerse varios sufijos encadenados
- Dentro de esas subcarpetas se colocarán ficheros especializados con el mismo nombre que el que tienen en la carpeta básica (ej: `strings.xml`)
- La aplicación al usar un recurso elegirá de entre los disponibles aquel en el encajen más sufijos.

Cambios de configuración en tiempo de ejecución

- Una aplicación puede responder a ciertos cambios de configuración mientras se ejecuta.
- Si lo hace, debe especificarse en el Manifiesto de la aplicación:

```
<activity android:name=".TodoList"
          android:label="@string/app_name"
          android:theme="@style/TodoTheme"
          android:configChanges="orientation|keyboardHidden" />
```

- Cuando se produzca el cambio se invocará el método `onConfigurationChanged` en la Actividad:

```
@Override
public void onConfigurationChanged(Configuration _newConfig) {
  super.onConfigurationChanged(_newConfig);
  [ ... Update any UI based on resource values ... ]
  if (_newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
    [ ... React to different orientation ... ]
  }
  if (_newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
    [ ... React to changed keyboard visibility ... ]
  }
}
```

Utilización de recursos en el código (I)

- Se accede a los recursos de una aplicación a través de la clase estática `R`, regenerada a partir de los ficheros XML cada vez que se recompila el proyecto.
- La clase `R` incluye subclases estáticas para cada uno de los tipos de recursos para los que se ha definido al menos un recurso. Ej: `R.string`, `R.drawable`
- Cada subclase expone sus recursos asociados como variables, con nombre igual a los identificadores de los recursos. Ej: `R.string.app_name`, `R.drawable.icon`
- El valor de estas variables es una referencia al recurso, no una instancia del recurso.
- Cuando un constructor o un método (como `setContentView`) acepta como parámetro un identificador de recurso, se le puede pasar una de estas variables:

```
1  // Inflate a layout resource.
   setContentView(R.layout.main);
3  // Display a transient dialog box that displays the
   // error message string resource.
5  Toast.makeText(this, R.string.app_error, Toast.LENGTH_LONG).show();
```

Utilización de recursos en el código (II)

- La tabla de recursos de una aplicación está representada por una instancia de la clase `Resources`.
- Cuando se necesita una instancia del recurso se usan métodos *helper* para extraerlos de la tabla de recursos.
- La clase `Resources` incluye *getters* para cada tipo de recursos disponible, a los que se pasa la variable (referencia) del recurso del que se quiere una instancia:

```
1  Resources myResources = getResources();
   CharSequence styledText = myResources.getText(R.string.stop_message);
3  Drawable icon = myResources.getDrawable(R.drawable.app_icon);

5  int opaqueBlue = myResources.getColor(R.color.opaque_blue);

7  float borderWidth = myResources.getDimension(R.dimen.standard_border);

9  Animation tranOut;
   tranOut = AnimationUtils.loadAnimation(this, R.anim.spin_shrink_fade);
11
   String[] stringArray;
13 stringArray = myResources.getStringArray(R.array.string_array);

15 int[] intArray = myResources.getIntArray(R.array.integer_array);
```

Referenciación de Recursos en otros Recursos

- Para referenciar un recurso en otro recurso se utiliza la notación @:

```
atributo="@[nombre_paquete:]tipo_recurso/id_recurso"
```

(el nombre de paquete puede omitirse si es del mismo paquete)

- Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:padding="@dimen/standard_border">
  <EditText
      android:id="@+id/myEditText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/stop_message"
      android:textColor="@color/opaque_blue"
  />
</LinearLayout>
```

Referenciación de Recursos del Sistema

- Las aplicaciones nativas de Android externalizan muchos de sus recursos para que sean accesibles desde otras aplicaciones.
- Para acceder a ellos en el código Java se utiliza `android.R` en vez de `R`:

```
1 CharSequence httpError = getString(android.R.string.httpErrorBadUrl);
```

- Para acceder desde otros recursos, se utiliza `android` como nombre de paquete:

```
1 <EditText
      android:id="@+id/myEditText"
3     android:layout_width="fill_parent"
      android:layout_height="wrap_content"
5     android:text="@android:string/httpErrorBadUrl"
      android:textColor="@android:color/darker_gray"
7 />
```

- También es posible acceder a recursos del tema actual (para mantener consistencia con el resto de aplicaciones), con `?`:

```
1     android:textColor=?android:textColor"
```

Contenidos

Terminología

- Vistas *(Views)*: Clase básica que representa a un elemento básico de IU en Android. En otros tipo de aplicaciones se utiliza el nombre de controles o *widgets*.
- Todos los componentes visuales en Android descienden de la clase `View`.
- Grupos de Vistas *(ViewGroups)*: Extensiones de la clase `View` para crear controles compuestos que contienen múltiples *Views* hijas.
- Diseños *(Layouts)*: Extensiones de la clase `ViewGroup` para gestionar la disposición de sus *Views* hijas.

Creación de la IU de una Actividad con *Views*

- La forma habitual es establecer el *Layout* al principio del método `onCreate`.
- Una vez establecido, pueden obtenerse referencias a las *Views* contenidas en el *Layout* llamando al método `findViewById`:

```
1  @Override
   public void onCreate(Bundle savedInstanceState) {
3    super.onCreate(savedInstanceState);

5    setContentView(R.layout.main);

7    TextView myTextView = (TextView)findViewById(R.id.myTextView);
```

Views estándar habituales

- `TextView`: Etiqueta de texto de solo lectura.
- `EditText`: Caja de texto editable, que acepta entradas multilínea.
- `ListView`: *ViewGroup* que gestiona un grupo de *Views* para mostrarlo en forma de lista, usando por defecto un *TextView* para cada elemento.
- `Spinner`: Control compuesto que muestra un *TextView* con una *ListView* asociada para elegir uno de sus elementos para ser mostrado en el *TextView*.
- `Button`: Botón pulsable estándar.
- `Checkbox`: Botón de dos estados representado por un caja marcada o desmarcada.
- `RadioButton`: Varios botones de dos estados, agrupados de forma que sólo uno del grupo puede estar activado.

Contenidos

Layouts estándar habituales

- `FrameLayout`: Coloca cada *View* en la esquina izquierda. Cada nueva *View* que se añade tapa a la anterior.
- `LinearLayout`: Coloca cada *View* hija en línea, de forma horizontal (una fila de *Views*) o vertical (una columna de *Views*).
- `TableLayout`: Coloca las *Views* en forma de tabla
- `RelativeLayout`: Las posiciones de cada *View* hija es relativa a las otras y a los bordes de la pantalla
- `AbsoluteLayout`: Las posiciones de cada *View* hija se define en coordenadas absolutas

Ejemplo:

```
1  <?xml version="1.0" encoding="utf-8"?>
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical"
      android:layout_width="fill_parent"
5     android:layout_height="fill_parent">
     <TextView
7        android:layout_width="fill_parent"
         android:layout_height="wrap_content"
9        android:text="Enter Text Below"
      />
11    <EditText
         android:layout_width="fill_parent"
13       android:layout_height="wrap_content"
         android:text="Text Goes Here!"
15    />
   </LinearLayout>
```

Contenidos

Creación de nuevas *Views* a partir de las existentes

- Se extiende una *View* ya existente, redefiniendo los métodos que definen su apariencia y su comportamiento:

MyTextView.java

```
public class MyTextView extends TextView {
  public MyTextView (Context context, AttributeSet ats, int defStyle) {
    super(context, ats, defStyle);
  }

  public MyTextView (Context context) {
    super(context);
  }

  public MyTextView (Context context, AttributeSet attrs) {
    super(context, attrs);
  }

  @Override
  public void onDraw(Canvas canvas) {
    [ ... Draw things on the canvas under the text ... ]

    // Render the text as usual using the TextView base class.
    super.onDraw(canvas);

    [ ... Draw things on the canvas over the text ... ]
  }

  @Override
  public boolean onKeyDown(int keyCode, KeyEvent keyEvent) {
    [ ... Perform some special processing ... ]
    [ ... based on a particular key press ... ]

    // Use the existing functionality implemented by
    // the base class to respond to a key press event.
    return super.onKeyDown(keyCode, keyEvent);
  }
}
```

Mejora del aspecto de la *TodoList* (I)

- Creamos una nueva `TodoListItemView` extendiendo `TextView`:

src/com.clau.todolist/TodoListItemView.java

```
1  package com.clau.todolist;
   import android.content.Context;
3  import android.content.res.Resources;
   import android.graphics.Canvas;
5  import android.graphics.Paint;
   import android.util.AttributeSet;
7  import android.widget.TextView;

9  public class TodoListItemView extends TextView {
     public TodoListItemView (Context context, AttributeSet ats, int ds) {
11     super(context, ats, ds);
       init();
13   }

15   public TodoListItemView (Context context) {
       super(context);
17     init();
     }
19
     public TodoListItemView (Context context, AttributeSet attrs) {
21     super(context, attrs);
       init();
23   }
     private void init() {
25   }

27   @Override
     public void onDraw(Canvas canvas) {
29     // Use the base TextView to render the text.
       super.onDraw(canvas);
31   }
   }
```

Mejora del aspecto de la *TodoList* (II)

- Creamos un nuevo `colors.xml` en `res/values`:

res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="notepad_paper">#AAFFFF99</color>
    <color name="notepad_lines">#FF0000FF</color>
    <color name="notepad_margin">#90FF0000</color>
    <color name="notepad_text">#AA0000FF</color>
</resources>
```

- Creamos un nuevo `dimens.xml` en `res/values`:

res/values/dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="notepad_margin">30px</dimen>
</resources>
```

Mejora del aspecto de la *TodoList* (III)

- Ahora podemos incluir el método `init` en `TodoListItemView`:

src/com.clau.todolist/TodoListItemView.java (fragmento)

```
    private Paint marginPaint;
2   private Paint linePaint;
    private int paperColor;
4   private float margin;

6   private void init() {
      // Get a reference to our resource table.
8     Resources myResources = getResources();

10    // Create the paint brushes we will use in the onDraw method.
      marginPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
12    marginPaint.setColor(myResources.getColor(R.color.notepad_margin));
      linePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
14    linePaint.setColor(myResources.getColor(R.color.notepad_lines));

16    // Get the paper background color and the margin width.
      paperColor = myResources.getColor(R.color.notepad_paper);
18    margin = myResources.getDimension(R.dimen.notepad_margin);
    }
```

Mejora del aspecto de la *TodoList* (IV)

- Refinamos el método `onDraw` utilizando las variables de instancia inicialidadas en `init`:

src/com.clau.todolist/TodoListItemView.java (fragmento)

```
1   @Override
    public void onDraw(Canvas canvas) {
3     // Color as paper
      canvas.drawColor(paperColor);
5
      // Draw ruled lines
7     canvas.drawLine(0, 0, getMeasuredHeight(), 0, linePaint);
      canvas.drawLine(0, getMeasuredHeight(),
9     getMeasuredWidth(), getMeasuredHeight(),
      linePaint);
11
      // Draw margin
13    canvas.drawLine(margin, 0, margin, getMeasuredHeight(), marginPaint);

15    // Move the text across from the margin
      canvas.save();
17    canvas.translate(margin, 0);

19    // Use the base TextView to render the text.
      super.onDraw(canvas);
21    canvas.restore();
    }
```

Mejora del aspecto de la *TodoList* (V)

- Creamos un nuevo *Layout* para especificar cómo se dispondrán los elementos de la lista usando la nueva *View*:

res/layout/todolist_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.clau.todolist.TodoListItemView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:scrollbars="vertical"
    android:textColor="@color/notepad_text"
    android:fadingEdge="vertical"
/>
```

Mejora del aspecto de la *TodoList* (VI)

- Ahora en el `onCreate` de `ToDoList.java` reemplazamos los parámetros pasados al `ArrayAdapter` para hacer uso del nuevo *Layout*.
- ANTES:

src/com.clau.todolist/TodoListItemView.java (fragmento)

```
final ArrayList<String> todoItems = new ArrayList<String>();
final ArrayAdapter<String> aa;
aa = new ArrayAdapter<String>(this,
  android.R.layout.simple_list_item_1, todoItems);
myListView.setAdapter(aa);
```

- AHORA:

src/com.clau.todolist/TodoListItemView.java (fragmento)

```
final ArrayList<String> todoItems = new ArrayList<String>();
int resID = R.layout.todolist_item;
final ArrayAdapter<String> aa;
aa = new ArrayAdapter<String>(this, resID, todoItems);
myListView.setAdapter(aa);
```

Contenidos

Creación de controles compuestos (I)

- Un Control Compuesto es aquel formado por múltiples *Views* que trabajan juntos.
- Al crearlo se define su disposición, apariencia y la interación entre sus *Views*.
- Se crean extendiendo un `ViewGroup` (o más bien un *Layout*):

MyCompoundView.java

```
1  public class MyCompoundView extends LinearLayout {
     public MyCompoundView(Context context) {
3      super(context);
     }
5    public MyCompoundView(Context context, AttributeSet attrs) {
       super(context, attrs);
7    }
   }
```

Creación de controles compuestos (II)

- Para diseñar su interfaz de usuario se utiliza un *Layout*:

clearable_edit_text.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText
        android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <Button
        android:id="@+id/clearButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Clear"
    />
</LinearLayout>
```

Creación de controles compuestos (III)

- Para asignar el *Layout* al nuevo control compuesto hay que redefinir su constructor:

MyCompoundView.java (fragmento)

```
1  EditText editText;
   Button clearButton;
3
   public ClearableEditText(Context context) {
5      super(context);

7      // Inflate the view from the layout resource.
       String infService = Context.LAYOUT_INFLATER_SERVICE;
9      LayoutInflater li;
       li = (LayoutInflater)getContext().getSystemService(infService);
11     li.inflate(R.layout.clearable_edit_text, this, true);

13     // Get references to the child controls.
       editText = (EditText)findViewById(R.id.editText);
15     clearButton = (Button)findViewById(R.id.clearButton);

17     // Hook up the functionality
       hookupButton();
19   }
   }
```

Contenidos

Tipos de Menú

- Menú de Opciones: El que aparece cuando se pulsa el botón "Menú" o "M" del terminal. Consta de un menú de iconos y, en su caso, de un menú expandido:
  - **Menú de Iconos**: Hasta 6 elementos con un icono y/o texto. En cada elemento puede ir un submenú. Si se definen más de 6 elementos, aparece un elemento llamado "Más" que mostrará el menú expandido.
  - **Menú Expandido**: Lista desplazable de los elementos del menú de opciones que no entraban dentro del menú de iconos.
- Menú Contextual: Menú que se despliega al pulsar el botón central del terminal, o al tocar prolongadamente la pantalla táctil.
- Submenú: Menú flotante que se despliega al seleccionar una entrada definida como submenú en alguno de los menús anteriores. Dentro de un submenú NO puede haber otro.

Creación de un Menú de Opciones para una Actividad

- Hay que redefinir el método `OnCreateOptionsMenu`, que se invocará la primera vez que se muestre el menú de opciones de la Actividad.
- En la implementación redefinida es necesario llamar al método de la clase `Activity`.
- Para añadir entradas al menú se utiliza el método `add` de la clase `Menu`, que requiere los siguientes parámetros:
  - un identificador de grupo para poder agrupar entradas
  - un identificador de la entrada, que permitirá saber qué entrada se ha pulsado cuando se invoque el manejador `onOptionsItemSelected`. Cada identificador de una entrada debería declararse como atributo de clase de la Actividad.
  - un número de orden para la colocación de la entrada en el menú
  - el texto de la entrada

```
static final private int MENU_ITEM = Menu.FIRST;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  // Group ID
  int groupId = 0;
  // Unique menu item identifier. Used for event handling.
  int menuItemId = MENU_ITEM;
  // The order position of the item
  int menuItemOrder = Menu.NONE;
  // Text to be displayed for this menu item.
  int menuItemText = R.string.menu_item;

  // Create the menu item and keep a reference to it.
  MenuItem menuItem = menu.add(groupId, menuItemId,
                               menuItemOrder, menuItemText);
  return true;
}
```

Elementos del Menú de Opciones

- La clase `MenuItem` permite añadir detalles adicionales a la entrada:
  - *Checkboxes* o *Radio Buttons*, sólo visibles en el menú expandido o en un submenú.
  - Atajos de teclado
  - Títulos abreviados
  - Iconos (sólo visibles en el menú de iconos)
  - Manejador a ejecutar cuando se seleccione este elemento (NO recomendado, es más eficiente usar `onOptionsItemSelected`)
  - Intenciones, que serán activadas cuando se seleccione este elemento.

Modificación dinámica del Menú de Opciones

- Puede modificarse un menú cada vez que se muestra redefiniendo el método `onPrepareOptionsMenu`:

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
  super.onPrepareOptionsMenu(menu);

  MenuItem menuItem = menu.findItem(MENU_ITEM);

  [ ... modify menu items ... ]

  return true;
}
```

Manejar las selecciones del menú

- A través del único manejador `onOptionsItemSelected`.
- El elemento seleccionado se recibe como parámetro `MenuItem`.
- Habrá que comparar su identificador con los establecidos al crear el menú:

```
public boolean onOptionsItemSelected(MenuItem item) {
  super.onOptionsItemSelected(item);

  // Find which menu item has been selected
  switch (item.getItemId()) {

    // Check for each known menu item
    case (MENU_ITEM):
      [ ... Perform menu handler actions ... ]
      return true;
  }

  // Return false if you have not handled the menu item.
  return false;
}
```

Creación de un Submenú

- A través de `addSubMenu`, que admite los mismos parámetros que add para añadir elementos normales.
- Los elementos de dentro del submenú se gestionan igual que los del menú de opciones.
- Adicionalmente se puede añadir un icono para el menú de opciones y para la cabecera del submenú cuando se muestre.

```
SubMenu sub = menu.addSubMenu(0, 0, Menu.NONE, "Submenu");
sub.setHeaderIcon(R.drawable.icon);
sub.setIcon(R.drawable.icon);

MenuItem submenuItem = sub.add(0, 0, Menu.NONE, "Submenu Item");
```

Menús Contextuales

- Puede crearse un menú contextual para una Actividad redefiniendo su método `onCreateContextMenu` y registrando la *View* que lo usará:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);

  menu.setHeaderTitle("Context Menu");
  menu.add(0, menu.FIRST, Menu.NONE,
           "Item 1").setIcon(R.drawable.menu_item);
  menu.add(0, menu.FIRST+1, Menu.NONE, "Item 2").setCheckable(true);
  menu.add(0, menu.FIRST+2, Menu.NONE, "Item 3").setShortcut('3', '3');
  SubMenu sub = menu.addSubMenu("Submenu");
  sub.add("Submenu Item");
}
```

- Las selecciones de un menú contextual se manejan de forma similar a las del menú de opciones, redefiniendo el método `onContextItemSelected` de la Actividad:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  super.onContextItemSelected(item);

  [ ... Handle menu item selection ... ]

  return false;
}
```

Menús para la *TodoList* (I)

- Creamos dos ficheros .png para iconos del menú: Uno con un signo "+" para añadir una nueva tarea (`add_new_item.png`), y otro con un aspa para borrar una tarea (`remove_item.png`).
- Copiamos las imágenes a la carpeta `res/drawable`.
- Modificamos el fichero `res/values/strings.xml` para definir los textos para los menús:

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">To Do List</string>
    <string name="add_new">Add New Item</string>
    <string name="remove">Remove Item</string>
    <string name="cancel">Cancel</string>
</resources>
```

Menús para la *TodoList* (II)

- Creamos un nuevo tema para la aplicación creando un nuevo fichero `res/values/styles.xml`, basado en un tema estándar de Android y fijando un tamaño del texto:

res/values/styles.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
   <resources>
3    <style name="ToDoTheme" parent="@android:style/Theme.Black">
       <item name="android:textSize">12sp</item>
5    </style>
   </resources>
```

- Aplicamos el tema a la Aplicación en el manifiesto:

AndroidManifest.xml

```
   <activity android:name=".ToDoList"
2            android:label="@string/app_name"
             android:theme="@style/ToDoTheme">
```

Menús para la *TodoList* (III)

- Añadimos constantes para definir los identificadores de los elementos del menú:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   static final private int ADD_NEW_TODO = Menu.FIRST;
    static final private int REMOVE_TODO = Menu.FIRST + 1;
```

Menús para la *TodoList* (IV)

- Redefinimos `onCreateOptionsMenu` para añadir los elementos del menú de opciones:

src/com.clau.todolist/TodoList.java (fragmento)

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);

  // Create and add new menu items.
  MenuItem itemAdd = menu.add(0, ADD_NEW_TODO, Menu.NONE,
                              R.string.add_new);
  MenuItem itemRem = menu.add(0, REMOVE_TODO, Menu.NONE,
                              R.string.remove);

  // Assign icons
  itemAdd.setIcon(R.drawable.add_new_item);
  itemRem.setIcon(R.drawable.remove_item);

  // Allocate shortcuts to each of them.
  itemAdd.setShortcut('0', 'a');
  itemRem.setShortcut('1', 'r');
  return true;
}
```

Menús para la *TodoList* (V)

- Para crear un menú contextual primero modificamos `onCreate` para indicar que la *ListView* tendrá dicho menú:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   @Override
     public void onCreate(Bundle onSaveInstanceState) {
3
    [ ... existing onCreate method ... ]
5
     registerForContextMenu(myListView);
7   }
```

- Ahora redefinimos `onCreateContextMenu`:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   @Override
    public void onCreateContextMenu(ContextMenu menu,
3                                   View v,
                                    ContextMenu.ContextMenuInfo menuInfo) {
5     super.onCreateContextMenu(menu, v, menuInfo);

7     menu.setHeaderTitle("Selected To Do Item");
      menu.add(0, REMOVE_TODO, Menu.NONE, R.string.remove);
9   }
```

Menús para la *TodoList* (VI)

- En el método `onCreate` necesitamos que la lista de items, el `ArrayAdapter` y los controles tengan visibilidad en otros métodos, así que los convertimos en atributos:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   private ArrayList<String> todoItems;
    private ListView myListView;
3   private EditText myEditText;
    private ArrayAdapter<String> aa;
5
    @Override
7   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
9
      // Inflate your view
11    setContentView(R.layout.main);

13    // Get references to UI widgets
      myListView = (ListView)findViewById(R.id.myListView);
15    myEditText = (EditText)findViewById(R.id.myEditText);

17    todoItems = new ArrayList<String>();

19    [ ... rest of existing onCreate method ... ]

21  }
```

Menús para la *TodoList* (VII)

- Redefinimos `onPrepareOptionsMenu` para que muestre "Cancel" en vez de "Delete" mientras se está añadiendo una entrada:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   private boolean addingNew = false;

3   @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
5     super.onPrepareOptionsMenu(menu);

7     int idx = myListView.getSelectedItemPosition();

9     String removeTitle = getString(addingNew ?
                                     R.string.cancel : R.string.remove);
11
      MenuItem removeItem = menu.findItem(REMOVE_TODO);
13    removeItem.setTitle(removeTitle);
      removeItem.setVisible(addingNew || idx > -1);
15
      return true;
17  }
```

Menús para la *TodoList* (VIII)

- Redefinimos `onOptionsItemSelected` en función de los métodos `cancelAdd`, `addNewItem` y `removeItem` que definiremos luego:

src/com.clau.todolist/TodoList.java (fragmento)

```
1  @Override
   public boolean onOptionsItemSelected(MenuItem item) {
3    super.onOptionsItemSelected(item);

5    int index = myListView.getSelectedItemPosition();

7    switch (item.getItemId()) {
       case (REMOVE_TODO): {
9        if (addingNew) {
           cancelAdd();
11       }
         else {
13         removeItem(index);
         }
15       return true;
       }
17     case (ADD_NEW_TODO): {
         addNewItem();
19       return true;
       }
21   }

23   return false;
   }
```

Menús para la *TodoList* (IX)

- Redefinimos `onContextItemSelected`:

src/com.clau.todolist/TodoList.java (fragmento)

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  super.onContextItemSelected(item);
  switch (item.getItemId()) {
    case (REMOVE_TODO): {
      AdapterView.AdapterContextMenuInfo menuInfo;
      menuInfo =(AdapterView.AdapterContextMenuInfo)item.getMenuInfo();

      int index = menuInfo.position;

      removeItem(index);
      return true;
    }
  }
  return false;
}
```

Menús para la *TodoList* (X)

- Definimos los métodos de apoyo `cancelAdd`, `addNewItem` y `removeItem`:

src/com.clau.todolist/TodoList.java (fragmento)

```
1   private void cancelAdd() {
      addingNew = false;
3     myEditText.setVisibility(View.GONE);
    }
5
    private void addNewItem() {
7     addingNew = true;
      myEditText.setVisibility(View.VISIBLE);
9     myEditText.requestFocus();
    }
11
    private void removeItem(int _index) {
13    todoItems.remove(_index);
      aa.notifyDataSetChanged();
15  }
```

Menús para la *TodoList* (XI)

- Dentro de `onCreate`, modificamos `onKeyListener` para incluir una llamada a `cancelAdd` después de añadir un item:

src/com.clau.todolist/TodoList.java (fragmento)

```
1  myEditText.setOnKeyListener(new OnKeyListener() {
     public boolean onKey(View v, int keyCode, KeyEvent event) {
3      if (event.getAction() == KeyEvent.ACTION_DOWN)
         if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER)
5        {
           todoItems.add(0, myEditText.getText().toString());
7          myEditText.setText("");
           aa.notifyDataSetChanged();
9          cancelAdd();
           return true;
11       }
       return false;
13   }
   });
```

Menús para la *TodoList* (XII)

- Modificamos el *layout* `res/layout/main.xml` para que la caja de texto esté oculta hasta que se elija la opción de añadir un nuevo elemento.

res/layout/main.xml (fragmento)

```
<EditText
    android:id="@+id/myEditText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:visibility="gone"
/>
```

Contenidos

Bibliografía

- Capítulos 3 y 4 de *Professional Android Application Development*. Reto Meier. Ed. Wrox, 2009.
- Capítulos 2 y 3 de *Hello, Android. Introducing Google's Mobile Development Platform*. Ed Burnette. Ed. The Pragmatic Bookshelf, 2009.
- Documentación del Android SDK: en la carpeta `docs` del directorio del SDK, o en `http://developer.android.com/guide/index.html`
- Documentación sobre Android (tutoriales, vídeos,...): `http://developer.android.com`