

PDM (DAM). IES El Majuelo



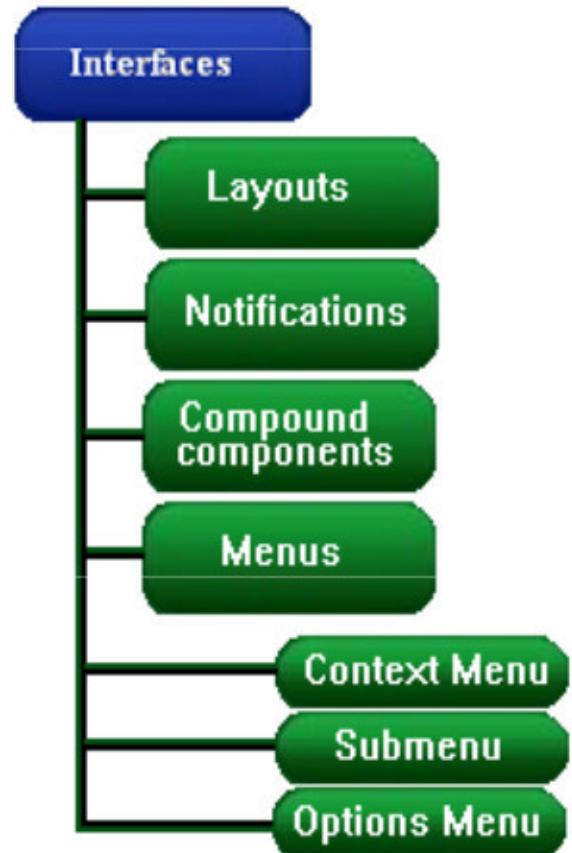
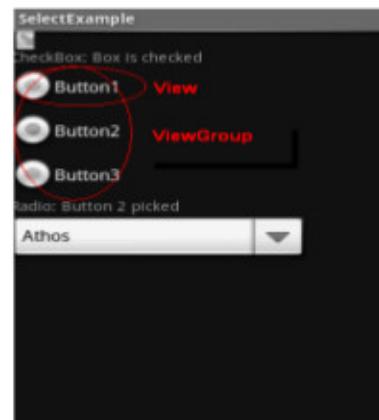
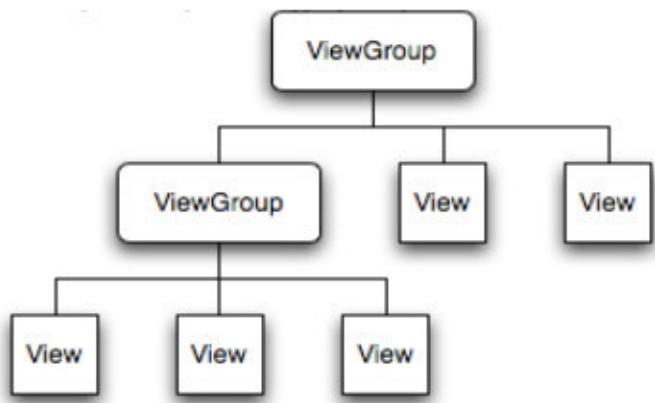
Jose Manuel Fornés Rumbao (copyrigh)
Departamento de Ingeniería Telemática
Universidad de Sevilla
jforres@us.es

Índice

1. Introducción
2. Views
3. Widgets
4. Layouts
 1. Tipos
 2. Captura de eventos
 3. Cambios de orientación
5. Notificaciones al usuario
6. Componentes compuestos
7. Menús
 1. Menú de opciones
 2. Menú contextual
 3. Submenús
8. Estilos y temas
9. Fragmentos
10. Interfaces avanzadas
 1. SurfaceView
 2. ViewFlipper
 3. ViewSwitcher

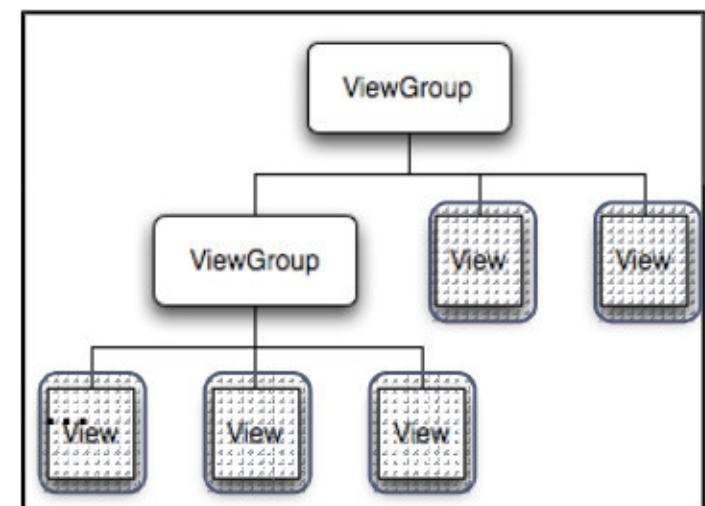
Introducción

- En Android, las interfaces se construyen utilizando los objetos `View` y `ViewGroup`. Todos los Views se colocan dentro de `GroupView`
- Las `Views` son el punto de interacción con el usuario (`setContentView()` en `Activity`)
- Las `ViewGroup` son una agrupación lógica de elementos visuales (`Views`)
- `android.widget` = subclases `View` con



Views

- Unidad básica de componente de UI
- Todas los controles visuales heredan de la clase `android.view.View`
- Ocupa un área rectangular en la pantalla.
- Son responsables de dibujarse:
 - Sus medidas, layout, como se pintan,...
- También gestionan los eventos = interacciones que reciben del usuario:
 - Cambio de focus, scrolling, clicks, gestos,
 - ...
- Se pueden crear por código (programáticamente) o por XML (declarativamente)



Views (programáticamente)

- NO recomendable ya que:
 - Creación de vista programática nos acoplamos a los detalles visuales del código.
 - Deberíamos repetir código para L10N, I18N, diferentes tamaños pantalla & pixel,
- ...

```
public void onCreate(Bundle icicle) {  
    super.onCreate(icicle);  
    TextView myTextView = new TextView(this);  
    myTextView.setText(Hola Clase);  
    setContentView(myTextView);  
}
```

Views (declarativamente –XML–)

- En el archivo /res/layout/main.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

- En el código de la Activity

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Widget (I)

- Son Views ‘preconstruidas’ que vienen incluidas en la plataforma Android.
- ¡NO confundir con App Widget (vistas en miniatura de las aplicaciones)!
- Están incluidos en el paquete android.widget.
- Son casi 50 clases y 30 interfaces
 - Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner ... AutoCompleteTextView, ImageSwitcher & TextSwitcher.
- Se pueden personalizar y crear widget nuevos.
 - Extender un subclase de View.
 - Implementar algunos override methods.
 - `onDraw()`, `onMeasure()` & `onKeyDown()`

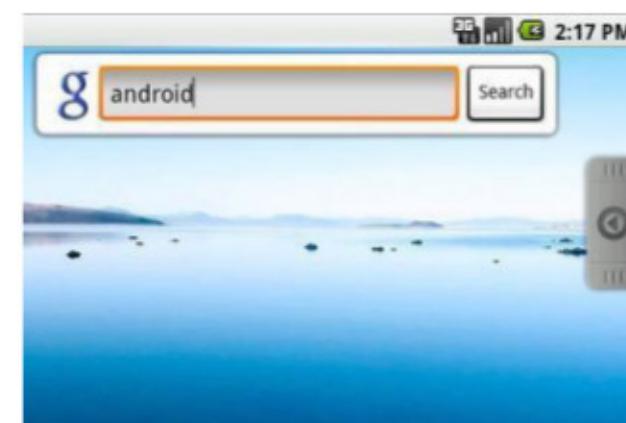
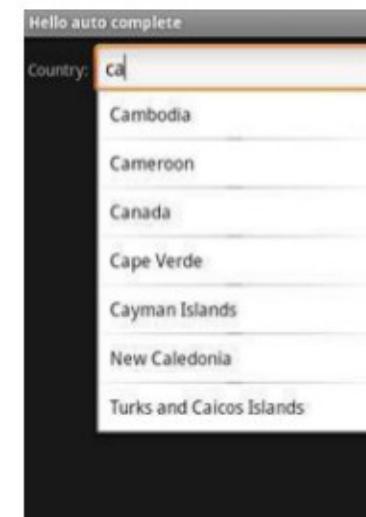


Widget (II)

- Views de texto
 - TextView: Muestra el texto, No permite editarlo (Label)
 - EditText: Componente de edición de texto. Acepta varias líneas
 - AutoCompleteTextView: Ofrece sugerencias del texto escrito
 - MultiAutoCompleteTextView: Ofrece sugerencias de cada palabra

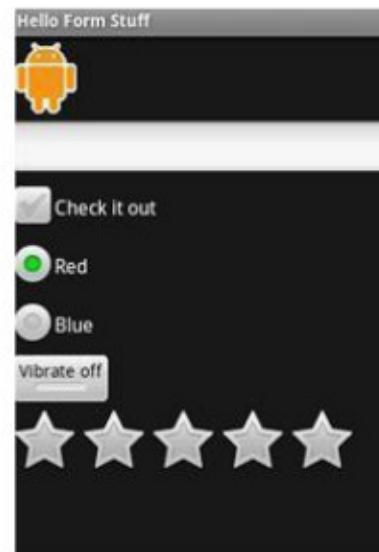
```
<TextView android:id="@+id/nameText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="name" />
```

```
TextView tv = (TextView)
findViewById(R.id.nameText);
tv.setText("name");
```



Widget (III)

- Views botones
 - Button: Botón estándar
 - ImageButton: Botón con imagen
 - ToggleButton: Botón de 2 estados ON/OFF
 - CheckBox: Checkbox estándar
 - RadioButton: RadioButton estandar. Selección simple



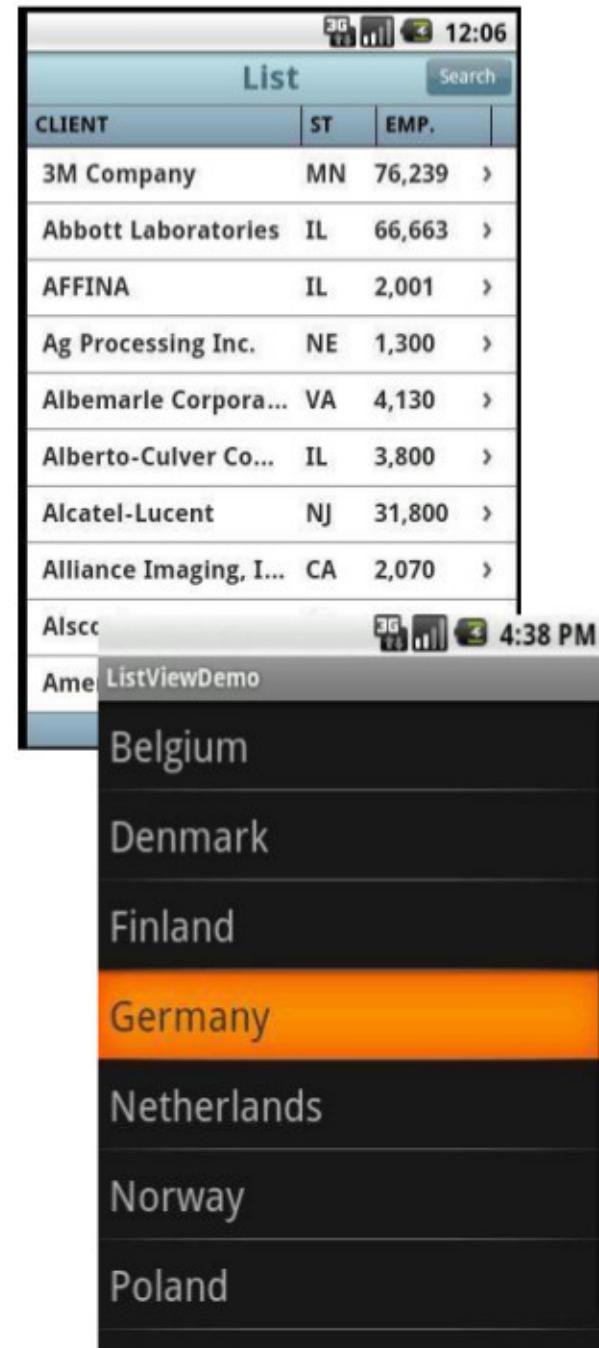
```
<ImageButton android:id="@+id/imageButton"
    android:src="@drawable/imageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```
ImageButton btn = (ImageButton)
findViewById(R.id.image);
btn.setImageResource(R.drawable.image);
```



Widget (IV)

- Views Listas
 - Muestra una lista de items verticalmente. Se puede diseñar la apariencia de los elementos de la lista. Normalmente se usa con una ListActivity y un ListAdapter
 - Los pasos para crearlo son:
 - Diseñar la vista de los elementos de la lista en el fichero layout correspondiente
 - Crear una actividad que herede de ListActivity
 - Pasarle los datos al adaptador de la lista
 - En onCreate, asignarle el ListAdapter a la actividad
 - Añadirle la vista a la actividad



Widget (V)

- Views Listas

```
<TextView android:id="@+id/listItem"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
```

```
public class HelloListView extends ListActivity
    private String[] fruits = {"Orange", "Apple", "Pear"};
    private ArrayAdapter[] adp = null;
    public void onCreate(Bundle savedInstanceState) {
        adp = new ArrayAdapter(this,R.id.listItem,
            fruits); setListAdapter(adp);
        setContentView(R.layout.listView);
    }
}
```

Widget (VI)

- Views WebView
 - Permite embeber un navegador en una actividad
 - Los pasos para crearlo son:
 - Diseñar la vista en el fichero layout correspondiente
 - Modificar el `AndroidManifest.xml` para añadir permisos y configurar la pantalla completa
 - Crear una actividad
 - En `onCreate`, llamar al `setContentView` pasándole el id de la vista



Widget (VII)

- Views **WebView**

```
<WebView android:id="@+id/webview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

```
<uses-permission android:name="android.permission.INTERNET" />
<activity android:name=".webView" android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar">
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    webView = (WebView) findViewById(R.id.webview);
    webView.loadUrl("http://www.google.es");
}
```

Widget (VIII)

- Views ImageView
 - Muestra una imagen arbitraria. Puede ser un ícono. Puede cargar la imagen tanto de los recursos como de un ContentProvider. Ofrece el control según el tipo de pantalla y permite el escalado y posicionamiento



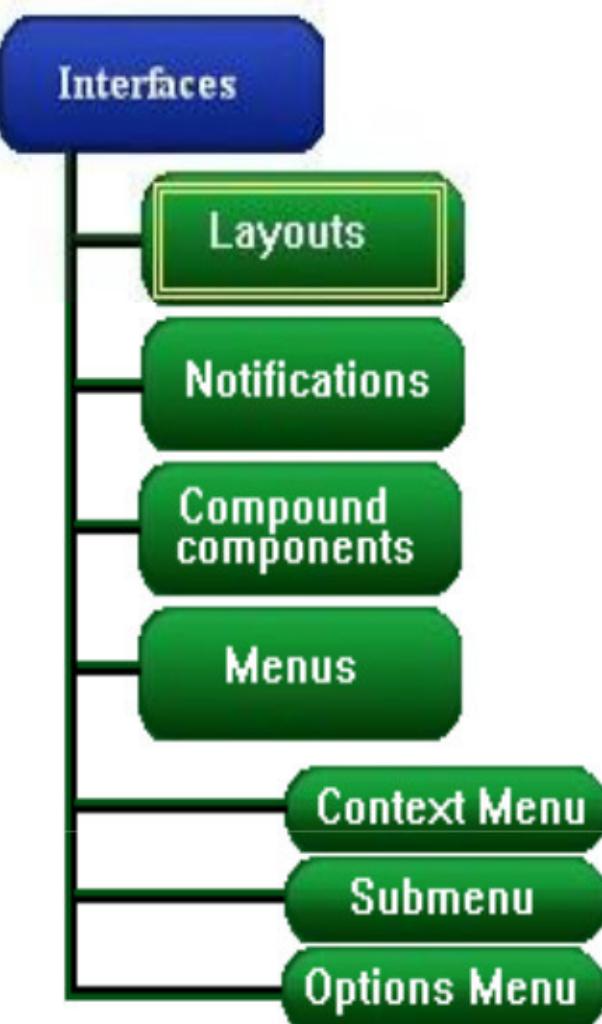
Widget (IX)

- Views ImageView

```
<ImageView  
    android:id="@+id/imageView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/android" />  
  
public class MyAndroidAppActivity extends Activity {  
  
    Button button;  
    ImageView image;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        addListenerOnButton();  
    }  
    public void addListenerOnButton() {  
        image = (ImageView) findViewById(R.id.imageView1);  
        button = (Button) findViewById(R.id.btnChangeImage);  
        button.setOnClickListener(new OnClickListener() {  
            @Override  
            public void onClick(View arg0) {  
                image.setImageResource(R.drawable.android3d);  
            }  
        });  
    }  
}
```

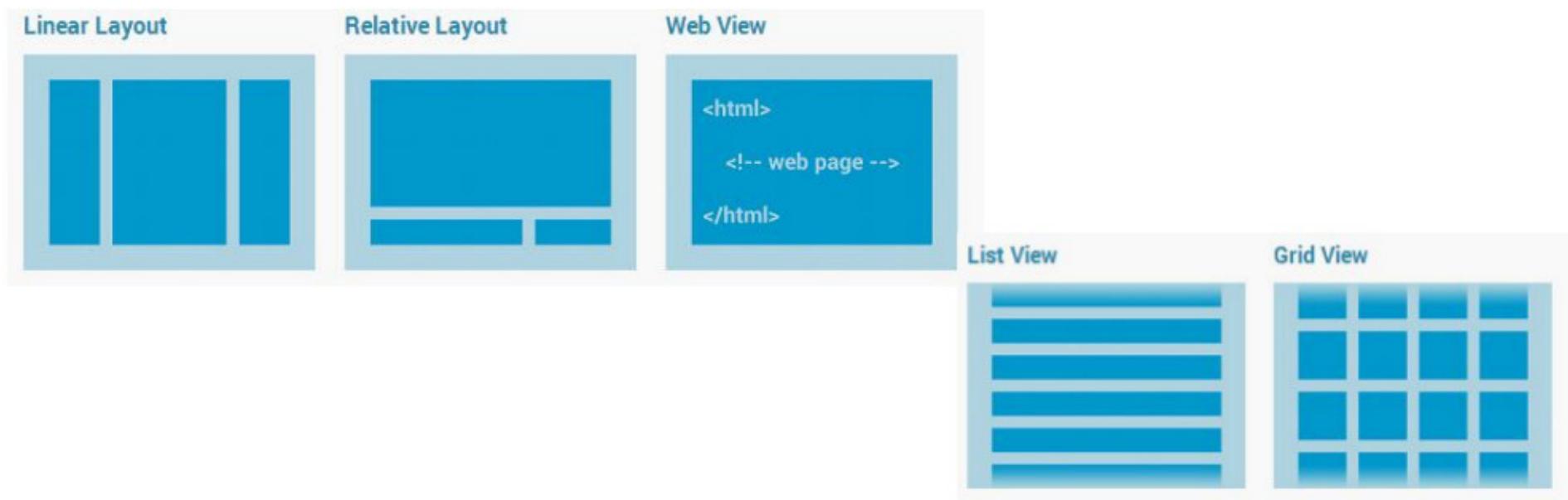
Layouts

- Su objetivo es controlar la posición de las Views hijas en la pantalla.
- Organizados jerárquicamente.
- Y También conocidos como Layout Managers.
- Son extensiones de ViewGroup
- Se pueden anidar, es decir, incluir Layouts dentro de Layouts
- Android proporciona una serie de Layouts por defecto



Layouts (tipos)

- FrameLayout: el más simple, añade cada View hija en la esquina superior izquierda. Cada vez que se añade una, tapa la anterior, diseñado para mostrar un único elemento.
- LinearLayout: añade cada View hija en línea recta (horizontal o vertical)
 - TableLayout: añade las Views usando un grid (cuadrícula), a través de filas y columnas. Es similar a las tablas HTML.
- RelativeLayout: añade las Views unas en relación a otras.
- AbsoluteLayout: añade las vistas dando coordenadas absolutas



Layouts (ID)

- Los Ids en los layouts y las Views son necesarias para referirnos a ellos tanto en el fichero XML de los layouts como en el código JAVA cuando queramos cambiar las Views:
 - Referencia en el código

```
<?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:text="Hi there"
        android:id="@+id/GreetingText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </EditText>
    <Button android:text="Push Me"
        android:id="@+id>HelloButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="buttonPushed">
    </Button>
</LinearLayout>
```

```
public void buttonPushed(View v) {
    EditText greetingText = (EditText) findViewById(R.id.GreetingText);
    greetingText.setText("Hello yourself");
}
```

Layouts (ID)

- Referencia en el layout XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <EditText android:text="Hi there"
        android:id="@+id/GreetingText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </EditText>
    <Button android:text="Push Me"
        android:id="@+id>HelloButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="buttonPushed"
        android:layout_toRightOf="@+id/GreetingText">
    </Button>
</RelativeLayout>
```

- Se usa @+ la primera vez que se utilice un `android:id` en el layout. El símbolo + le indica al aapt que este es un nuevo recurso y que por tanto debe añadir esta referencia

Layouts (unidades de medida)

- Cuando se especifica el tamaño de un elemento, podemos utilizar:
 - dp: pixel independiente de la densidad. 160p equivalen a 1 pulgada de pantalla física
 - sp: pixel independiente de la escala, se usa en tamaño de fuente de letra
 - pt: punto. Equivale a 1/72 pulgadas, basada en el tamaño de la pantalla
 - px: pixel. Equivale a los píxeles de la pantalla, no recomendable ya que no renderizará correctamente según la pantalla del terminal

Capturar eventos

- Hay más de una manera de interceptar los eventos de interacción del usuario:
 - Extender la clase View y sobrecargar el método

```
public class LoginExampleImplements extends Activity implements  
OnClickListener {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        // Set Click Listener  
        btnLogin.setOnClickListener(this);  
        btnCancel.setOnClickListener(this);  
    }  
    @Override  
    public void onClick(View v) {  
        ...  
        finish();  
    }  
}
```

Capturar eventos (II)

- EventListener relativos a cada View o ViewGroup. Cada evento tendrá una sólo un método Callback que será llamado cuando el usuario interactúa con la vista:
 - onClick, onLongClick, onTouch, etc.
- EventHandler, eventos del dispositivo. No importa dónde está el foco. No están necesariamente asociados a una vista:
 - onKeyDown, onKeyUp, onTouchEvent

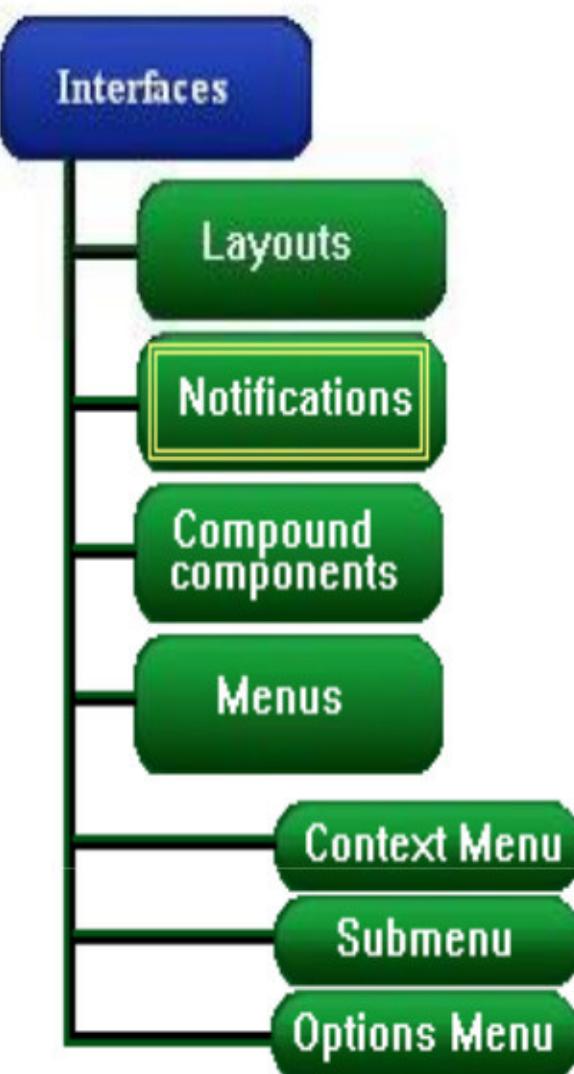
```
public class EventActivity extends Activity implements  
OnClickListener {  
    private View view;  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        view = (View) findViewById(R.id.view);  
        view.setOnClickListener(this);  
    }  
}
```

- Android recomienda que la actividad implemente la interfaz onXXXListener en lugar de usar clases anónimas

Cambios de orientación

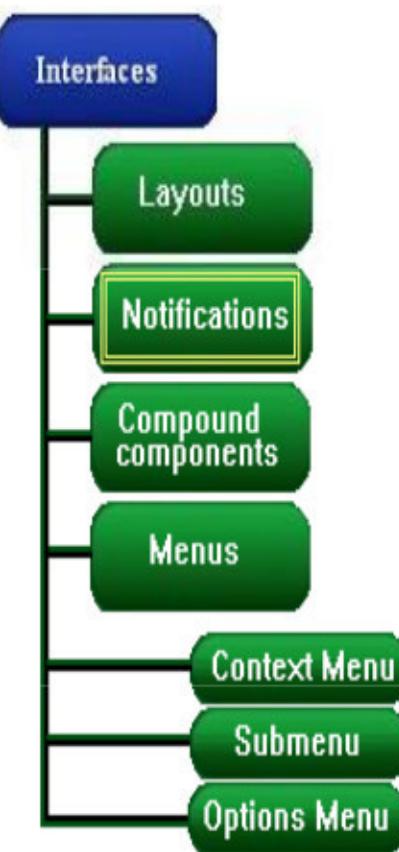
- Cuando cambia la orientación del dispositivo, la Activity visible se destruye y se vuelve a crear. Tanto la que se estaba ejecutando como las que estaban pausadas. Esto ocurre también con cualquier cambio de configuración como el idioma.
- Se debería volver a visualizar lo mismo que estaba antes del cambio, incluyendo lo que estuviera seleccionado
- Para ello usamos la persistencia de la actividad con los métodos onSaveInstanceState y onRestoreInstanceState
- Una vez que tenemos los datos nos reposicionamos en función de las nuevas dimensiones y nuevos layouts

Notificaciones al usuario (I)



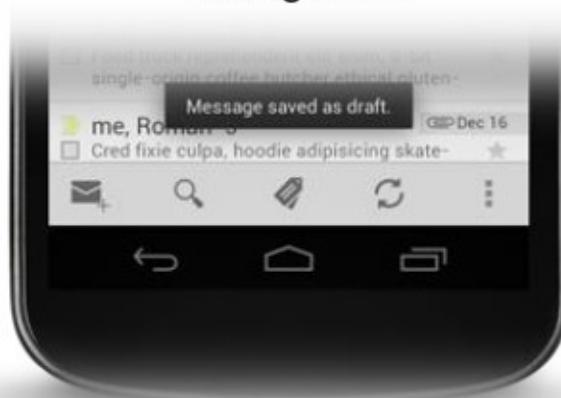
- Una vez añadido View/Widget a nuestro UI, necesitamos conocer como interaccionar/notificar al usuario.
- Algunas notificaciones requieren respuesta del usuario, otras no.
- Ejemplos: batería baja, confirmación de recepción de un archivo, barras de progreso, etc.
- Para cada tipo de notificación se requerirá una manera concreta de informar al usuario: vibración, iluminación LEDs

Notificaciones al usuario (II)



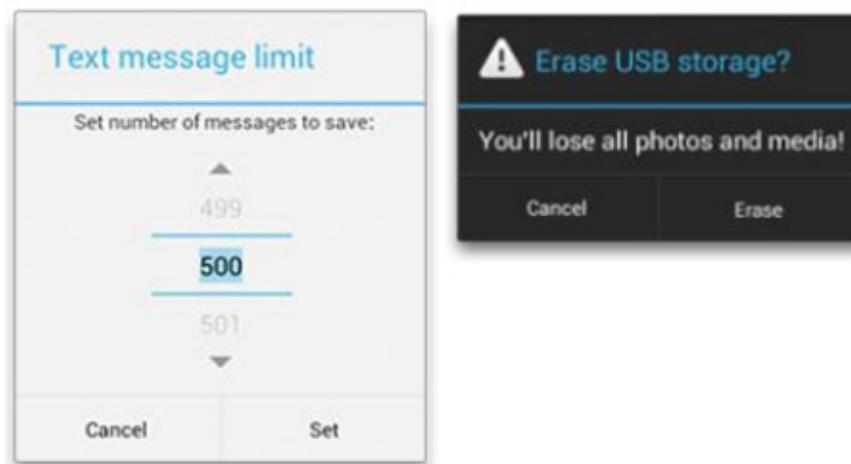
Toasts

Para breves mensajes desde el background



Dialog

Para notificaciones relacionadas con la Activity

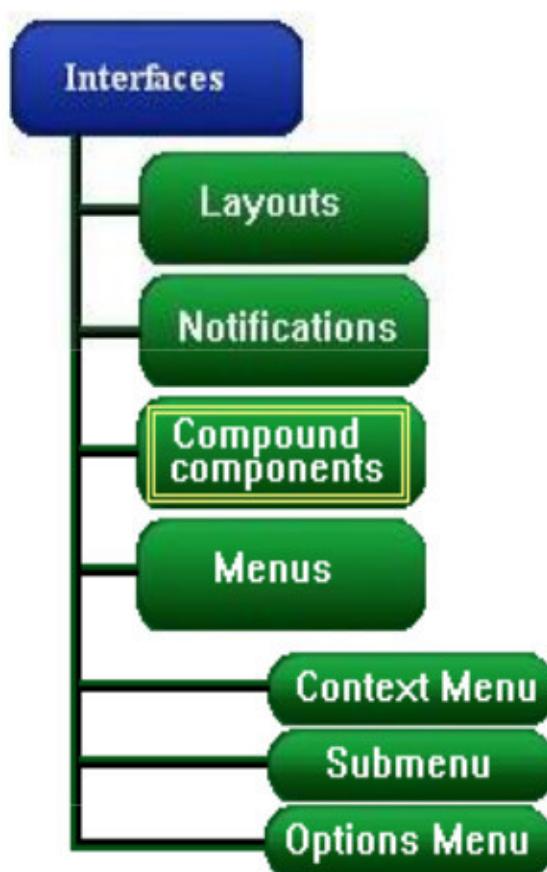


Status Bar

Para recordatorios persistentes que requieren del usuario



Componentes compuestos (I)



- Normalmente hacen referencia a Views con funcionalidad compleja añadida
- Para crear componentes compuestos, se pueden crear Views y añadirle métodos para la gestión de eventos, componentes visuales, etc
- Se pueden extender (heredar) componentes de Android por defecto, sobrescribiendo la funcionalidad que se necesite

Componentes compuestos (II)

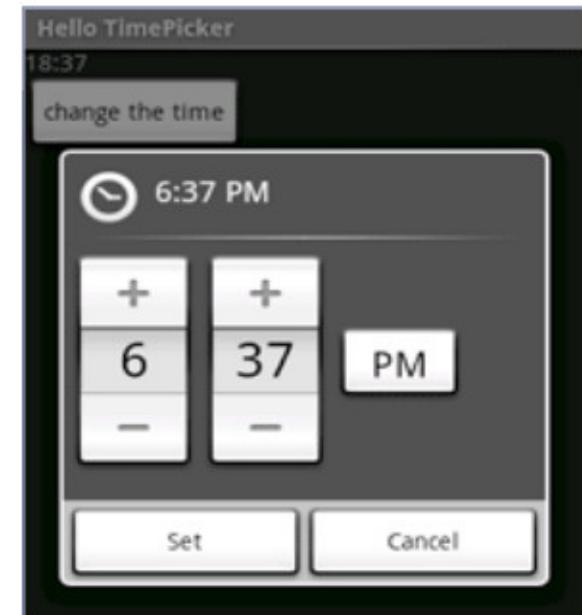
Spinner



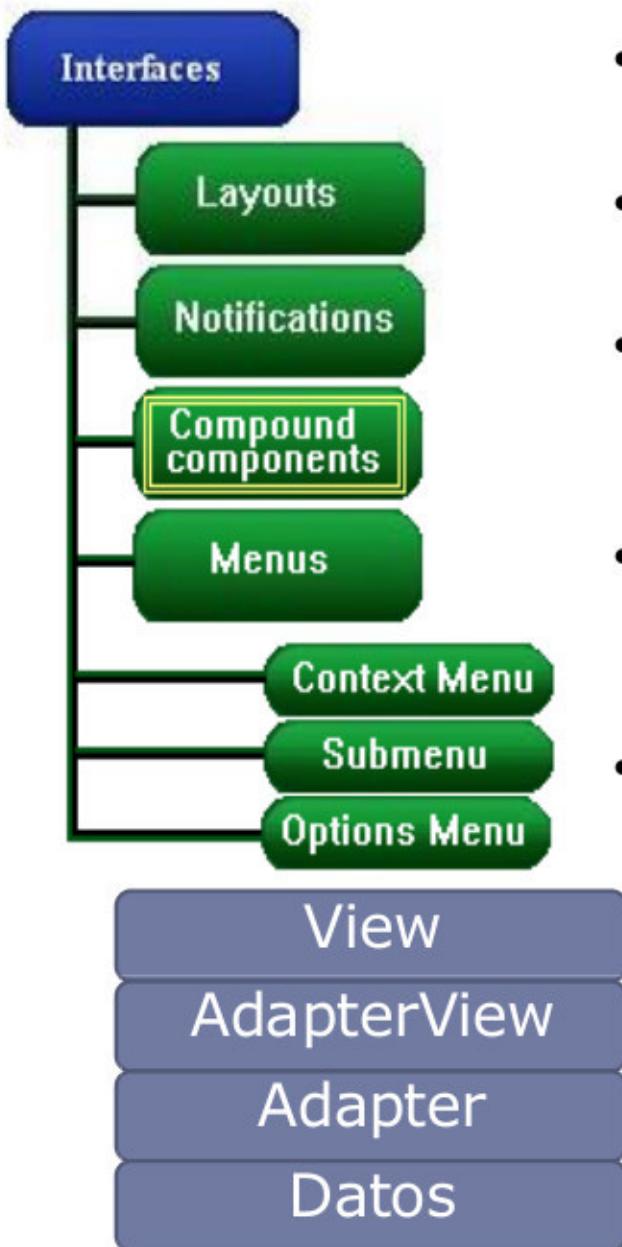
DatePicker



TimePicker



AdapterView



- Son Views cuyos hijos están determinados por un Adapter.
- Convierte la interfaz de una clase a otra que el cliente espera.
- Se usan para adaptar los datos a otro formato para que se puedan mostrar en una vista
- Encargados de llenar datos (binding) y gestionar selecciones de usuario (handling).
- Útiles cuando queremos mostrar visualmente datos almacenados

Button, ImageButton, EditText, ...
ListView, GridView, Spinner, Gallery, ...
CursorAdapter, ListAdapter, SpinnerAdapter,
Content Provider, Cursor, String {}, File, URI

Menus: Options Menu (I)



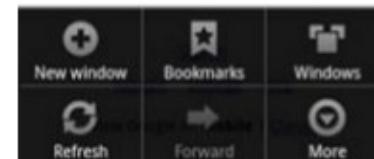
- Contiene un máximo de 6 elementos con iconos.
 - Si se necesitan más, se incluyen en el menú extendido (aparece al pulsar el botón +)
- El único que no soporta checkbox ni radio-buttons.
- La primera vez que se abra, Android llamará al método `onCreateOptionsMenu ()` de la Activity
- Cuando el usuario presione una opción del menú, se invocará el método `onOptionsItemSelected ()`

Menus: Options Menu (II)

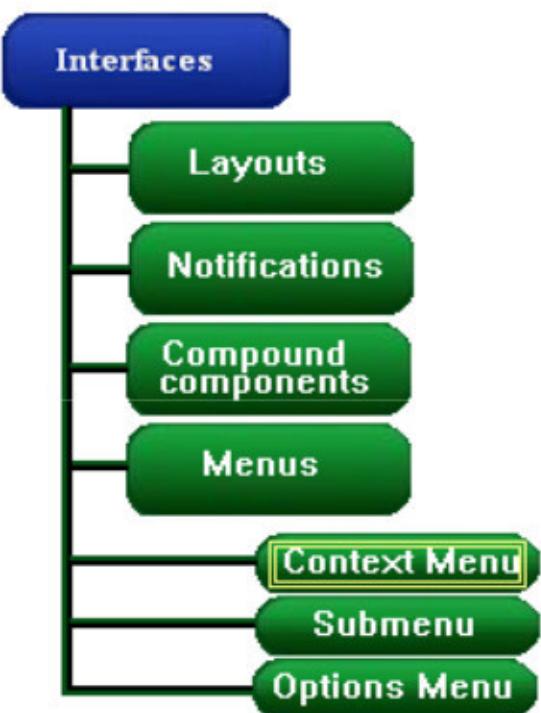
- Creación del Options Menu vía código

```
/* Creates the menu items */
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_NEW_GAME, 0, "New Game");
    menu.add(0, MENU_QUIT, 0, "Quit");
    return true;
}

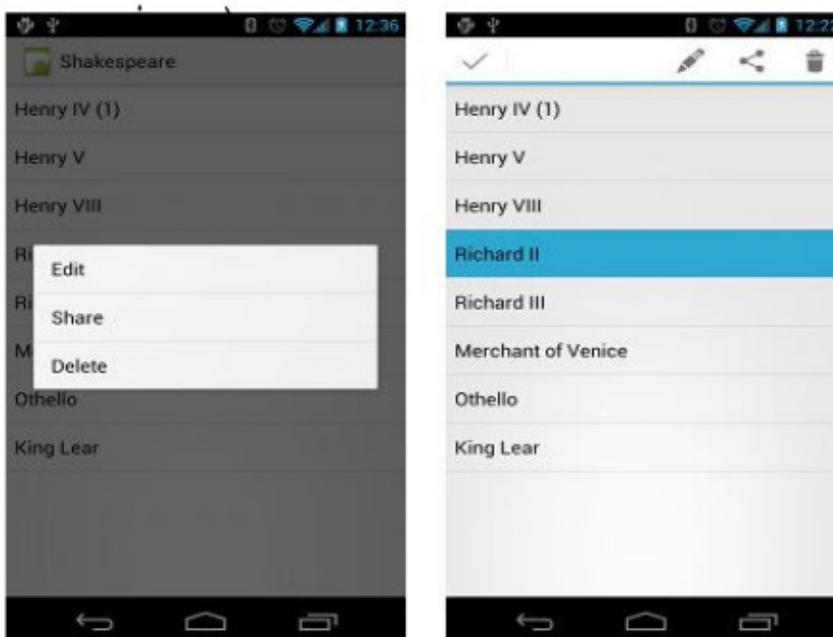
/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case MENU_NEW_GAME:
            newGame(); return
            true;
        case MENU_QUIT:
            quit();
            return true;
    }
    return false;
}
```



Menus: Context Menu (I)



- Conceptualmente similar al botón derecho del ratón en el PC.
- Presión sobre la vista unos dos segundos.
- Al aparecer, se invocará el método `onCreateContextMenu()`
- Al seleccionar, se invocará el método `onContextItemSelected()`
 - Para asociar una `View` a este menu
 - `registerForContextMenu (View`



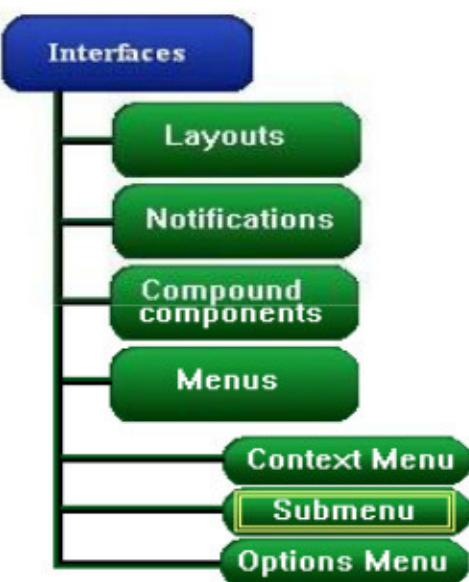
Menus: Context Menu (II)

- Creación del Context Menu vía código

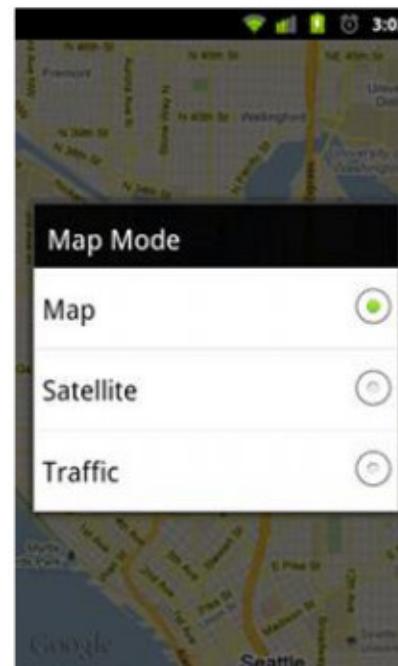
```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.add(0, EDIT_ID, 0, "Edit"); menu.add(0, DELETE_ID, 0,
        "Delete");
}
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case EDIT_ID:
            editNote(info.id);
            return true;
        case DELETE_ID:
            deleteNote(info.id);
            return true; default:
            return super.onContextItemSelected(item);
    }
}
```



Menus: Submenu



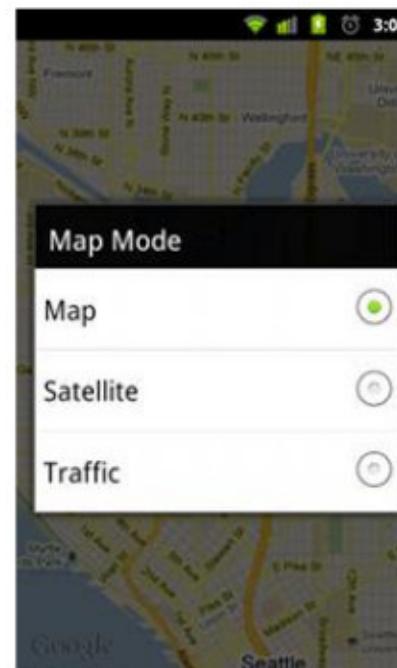
- Se puede añadir dentro de cualquier menú, excepto en otro submenu.
- Ideal para categorizar funcionalidades de nuestra app.
- Igual que en los Option Menu, al pulsar de invocará el método
`onOptionsItemSelected()`



Menus: Submenu (II)

- Creación del Submenu vía código

```
public boolean onCreateOptionsMenu(Menu menu) {  
    boolean result = super.onCreateOptionsMenu(menu);  
  
    SubMenu fileMenu = menu.addSubMenu("File");  
    SubMenu editMenu = menu.addSubMenu("Edit");  
    fileMenu.add("new");  
    fileMenu.add("open");  
    fileMenu.add("save");  
    editMenu.add("undo");  
    editMenu.add("redo");  
  
    return result;  
}
```



Estilos y temas

- En Android el estilo se especifica en un archivo XML
- El **estilo** es una apariencia que se aplica a una vista
- El **tema** es un estilo que se aplica a una actividad o una aplicación

```
<TextView android:layout_width="fill_parent"  
         android:layout_height="wrap_content"  
         android:textColor="#00FF00"  
         android:typeface="monospace"  
         android:text="@string/hello" />
```

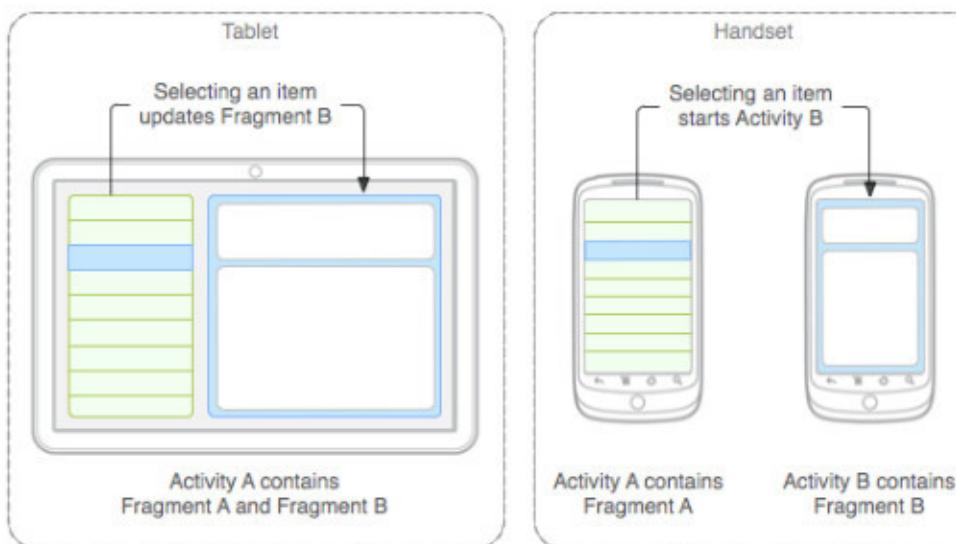
```
<TextView style="@style/CodeFont"  
         android:text="@string/hello" />
```

```
<resources>  
    <style name="CodeFont">  
        <item name="android:layout_width">fill_parent</item>  
        <item name="android:layout_height">wrap_content</item>  
        <item name="android:textColor">#00FF00</item>  
    </style>  
</resources>
```

- Cada hijo de resources es convertido en un recurso de la aplicación al compilar y puede ser referenciado desde un XML como p.e: @style/CodeFont

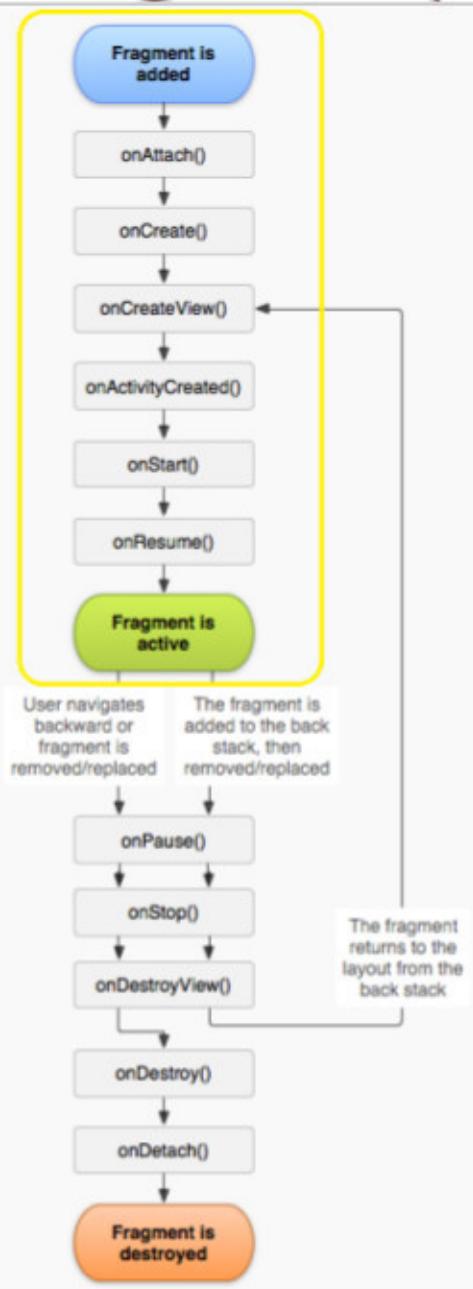
Fragments(I)

- Los Fragments se introdujeron en la versión de Android 3.0 para poder realizar código que se adapta los distintos tamaños de pantalla tan dispares como smartphones y tablets



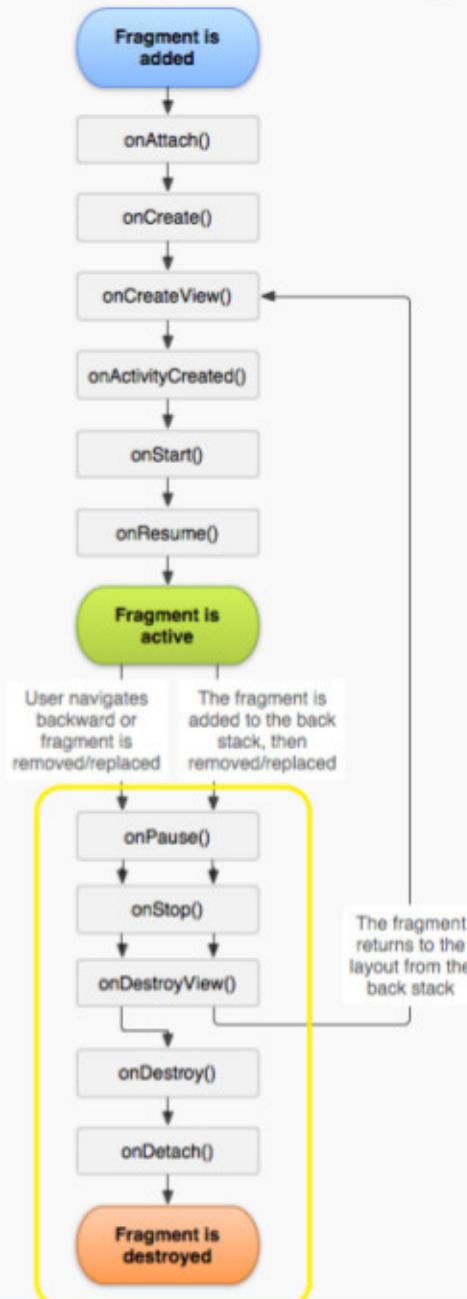
- Es un trozo (fragmento) de una Activity pero con su propio Layout y su propio ciclo de vida
- Siempre tiene que pertenecer a una Activity y no existe independientemente de ella

Fragments(II)



- El ciclo de vida de la Activity contenedora condiciona al del Fragment, p.e. si la Activity se pausa el Fragment también se pausará
 - El núcleo del ciclo de vida de un Fragment es:
 - `onAttach` (Activity) se llama una vez cuando el Fragment se asocia a su Activity
 - `onCreate (Bundle)` Se llama para la creación inicial del Fragment
 - `onCreateView (LayoutInflater, ViewGroup, Bundle)` crea y devuelve la jerarquía de vistas asociadas con el Fragment según su Layout
 - `onActivityCreated (Bundle)` dice al Fragment que su Activity ha completado su propio `Activity.onCreate()`.
 - `onViewStateRestored (Bundle)` dice al Fragment que todo el estado guardado de su jerarquía de vistas se ha restaurado
 - `onStart()` hace visible el Fragment al usuario (basado en que su Activity contenedora se ha iniciado).
 - `onResume()` permite la interacción del Fragment con el usuario (basado en que su Activity contenedora también lo ha hecho).

Fragments(III)



– Cuando un Fragment ya no se usa:

- **onPause()** El Fragment ya no interacciona con el usuario ya sea porque su Activity se ha pausado o porque una operación en el Fragment lo está modificando
- **onStop()** El Fragment ya no está visible al usuario ya sea porque su Activity se ha parado o porque una operación en el Fragment lo está modificando en la Activity
- **onDestroyView()** permite al Fragment liberar los recursos asociados con su View
- **onDestroy()** llamado para hacer la limpieza final del estado del Fragment
- **onDetach()** llamado inmediatamente antes de que un Fragment vaya a dejar de estar asociado a su Activity

Fragments(IV)

- Los Fragments se crean extendiendo la clase Fragment y sobrescribiendo el método `onCreateView()`, devolviendo la vista de dicho fragmento

```
public class FragmentUNO extends Fragment {  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        //Suponiendo que tenemos un layout llamado fragment_uno.xml  
        return inflater.inflate(R.layout.fragment_uno, container, false);  
    }  
}
```

- Se pueden crear los Fragments mediante subclases:
 - DialogFragment: Muestra un cuadro de diálogo flotante
 - ListFragment: Muestra una lista de elementos
 - PreferenceFragment: Muestra una lista de preferencias
 - WebViewFragment: Muestra una vista Web

Fragments(IV)

- Los Fragments se crean extendiendo la clase Fragment y sobrescribiendo el método `onCreateView()`, devolviendo la vista de dicho fragmento

```
public class FragmentUNO extends Fragment {  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        //Suponiendo que tenemos un layout llamado fragment_uno.xml  
        return inflater.inflate(R.layout.fragment_uno, container, false);  
    }  
}
```

- Se pueden crear los Fragments mediante subclases:
 - DialogFragment: Muestra un cuadro de diálogo flotante
 - ListFragment: Muestra una lista de elementos
 - PreferenceFragment: Muestra una lista de preferencias
 - WebViewFragment: Muestra una vista Web

Fragments(V)

- Una vez creados los Fragments con su layout, hay que agregarlos a una Activity:
 - Mediante Layout de la Activity, indicando el android:name como el paquete donde está ubicado el fragmento

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <fragment  
        android:name="com.dam.fragments.FRAGMENTOS.FragmentUNO"  
        android:id="@+id/fragment_uno"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"/>  
</LinearLayout>
```

- Mediante programación, indicando la id de una vista padre (ViewGroup) donde se colocará el Fragment, usando el método add() de la API FragmentTransaction. Se necesita un FrameLayout como lugar donde colocar el Fragment después

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.mostrar_fragment_dos);  
    FragmentManager FM = getSupportFragmentManager();  
    FragmentTransaction FT = FM.beginTransaction();  
    Fragment fragment = new FragmentUNO();  
    FT.add(R.id.fragment_container, fragment);  
    FT.commit();  
}
```

Interfaces avanzadas

- SurfaceView
 - Proporciona una superficie de dibujo dedicado incrustado dentro de una jerarquía de vistas. Básicamente es un View que contiene un Surface que es un buffer usado por Android cuando renderiza la pantalla.
 - Puede ser dibujado por varios hilos diferentes del de interfaz de usuario
 - No puede ser transparente pero puede estar detrás de otros elementos
 - Es más rápido y eficiente que un View. Uso adecuado para cámara.
 - Se puede definir en el Layout o programáticamente



Interfaces avanzadas

- ViewFlipper
 - Es un ViewAnimator que anima entre dos o más vistas que se han incorporado a él. Solo un hijo se muestra cada vez. Se puede poner que cambie entre hijos cada cierto tiempo de forma automática



Interfaces avanzadas

- ViewSwitcher
 - Es un ViewAnimator que intercambia entre dos Views y dispone de un mecanismo de creación de estas vistas. Las vistas se desplazan para ocultar y aparecer cada vez
 - Sólo se dispone de dos Views y una de ellas es la que se visualiza cada vez

