Tema 6. Diseño Avanzado de interfaces

Prof. Manuel Castillo

Programación de Dispositivos Móviles

Escuela Profesional de Ciencias de la Computación

Facultad de Ciencias

Universidad Nacional de Ingeniería

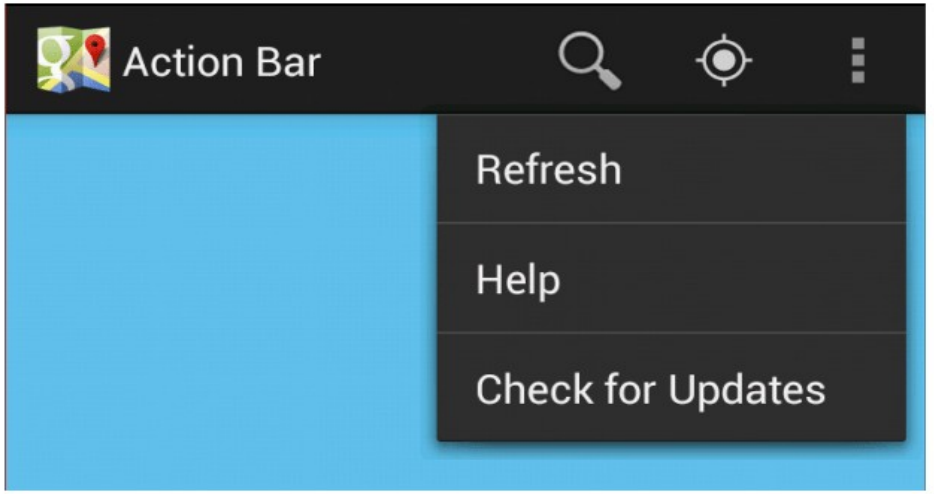
Objetivos

- Crear interfaces avanzadas.
- Saber lo que es un *NavegationView*, *ActionBar* y *Fragments*.
- Conocer los distintos recursos que podemos manejar en Android.
- Saber cómo adaptar las interfaces a las distintas pantallas.
- Saber cómo traducir las aplicaciones.

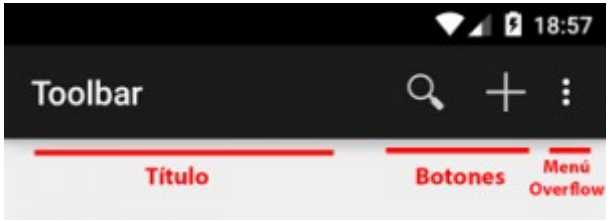
Índice de contenido

1. ActionBar

Action Bar adding Action Items



www.androidhive.info



1.1. Definición

- Soporte de una barra de navegación persistente en las aplicaciones.
- Intenta dar otra perspectiva a la antigua tecla de Menú.
- Coloca en la barra un icono de tres puntos que al pulsarlo despliega las opciones de Menú.
- Tiene un título en la barra.
- Coloca un icono principal de la aplicación en la barra.
- Permite colocar iconos de acceso directo en la barra principal.

1.2. Conceptos

- Icono: lo toma del manifiesto del atributo *android:icon* de la Activity.
- Título: lo toma del manifiesto del atributo *android:label* de la Activity.
- Menú: se describe en */res/menu/menu.xml* que cargamos con el *onCreateOptionsMenu* de la *Activity*. El fichero *XML* define una serie de items a mostrar:
  - *id*: identificativo.
  - *title*: título de la opción.
  - *icon*: icono de la acción.
  - *showAsAction*: marca si es una acción:
    - *never*: aparece dentro de las opciones de los tres puntos.
    - *ifRoom*: lo muestra si cabe en la barra sino dentro de los tres puntos.
    - *always*: lo mostrará siempre.
    - *withText*: lo mostrará con el título.

1.3. Fichero menú *XML*

```
<menu xmlns:android="http.....” ... tools:context=".MainActivity">
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:orderInCategory="100"
          app:showAsAction="never"/>
    <item android:id="@+id/action_buscar"
          android:title="@string/action_buscar"
          android:icon="@drawable/ic_menu_save"
          android:orderInCategory="100"
          app:showAsAction="ifRoom" />
    <item android:id="@+id/action_nuevo"
          android:title="@string/action_nuevo"
          android:icon="@drawable/ic_menu_add"
          android:orderInCategory="100"
          app:showAsAction="ifRoom|withText" />
</menu>
```

Lo muestra dentro de los tres puntos

Muestra si cabe

Combinación de dos opciones

1.4. Estilos

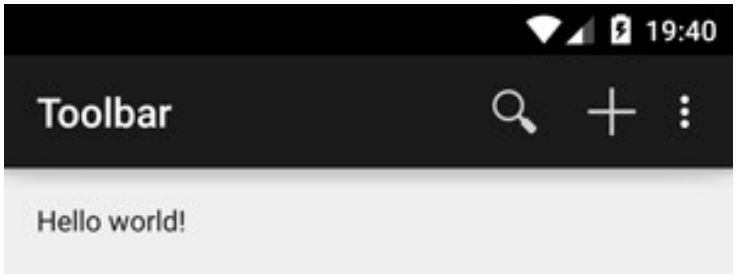
```
// En /res/values/style.xml
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme"
        parent='Theme.AppCompat.Light.DarkActionBar'>
            <item name="colorPrimary">@color/color_primary</item>
            <item name="colorPrimaryDark">@color/color_primary_dark
                </item>
            <item name="colorAccent">@color/color_accent</item>
    </style>
</resources>
// Crear un nuevo xml para los colores /res/values/colors.xml
<resources>
    <color name="color_primary">#3F51B5</color>
    <color name="color_primary_dark">#303F9F</color>
    <color name="color_accent">#FF4081</color>
</resources>
```

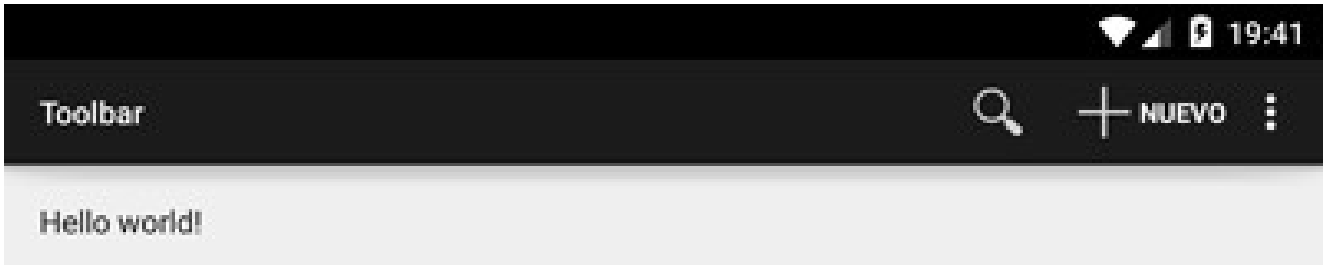
Color principal de la aplicación

Variante más oscura del color principal a partir de Android 5.0 para barra de estado (donde está el reloj)

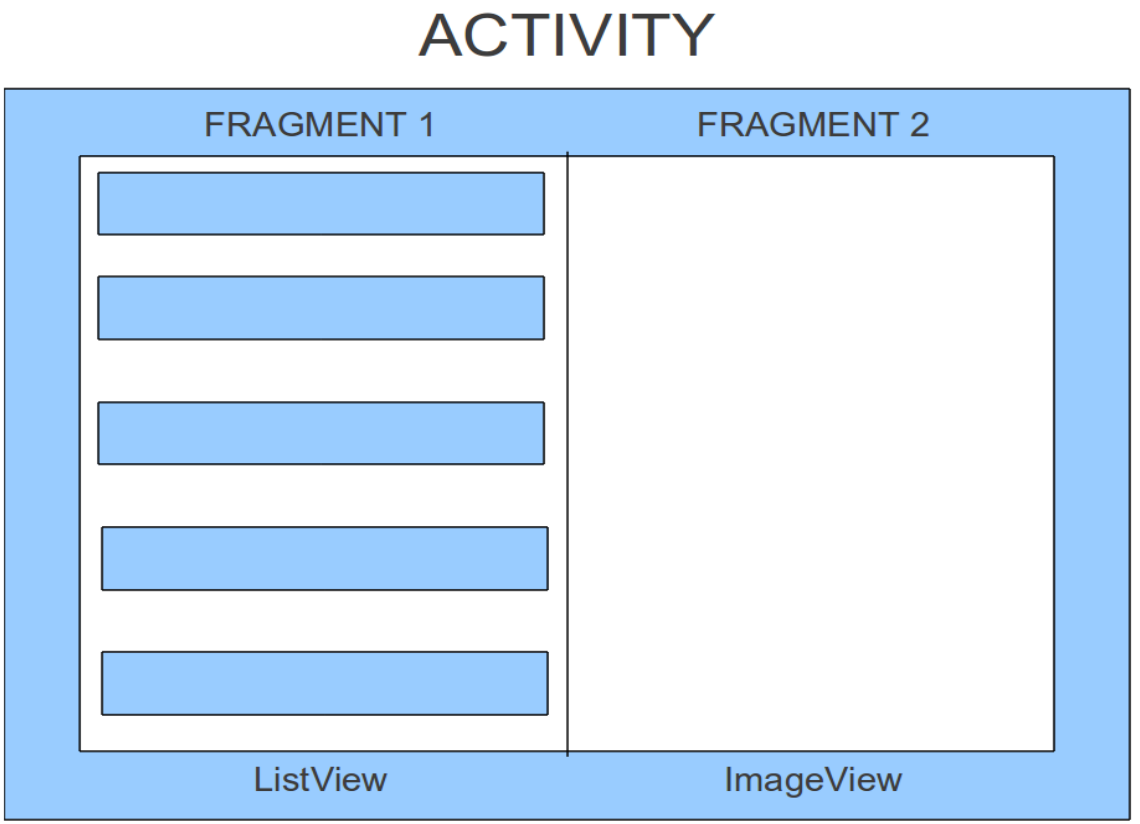
Color destacado para el botón de acción principal (por ejemplo botón flotante)

¿Que mostraría?





2. Fragments



2.1. Definición

- Son una serie de componentes visuales que se pueden presentar en distintas *Activities*.
- Tiene asociado normalmente un *Layout* y una clase Java.
- Desde las *Activities* podremos cargar dichos fragmentos.
- Las *Activities* que quieran usar fragmentos deberán heredar de *AppCompactActivity* (en versión anteriores de *FragmentActivity*).
- Las *Activities* tendrán que hacer uso del servicio que maneja los fragmentos antes de cargarlos en su propio *Layout*.

2.2. Implementación

- Se define una clase que hereda de *Fragment*.
- Se tiene que declarar un *Layout* para el fragmento con el contenido de vistas que se desee.
- En vez de tener un *onCreate* tiene un *onCreateView*.
  - Este método debe cargar el Layout del fragmento mediante un *Inflater* y devolverlo.
- En el *onStart* podemos comprobar si nos pasan argumentos a la llamada y actuar en consecuencia.
- Podemos interactuar con el *findViewById* pero de una manera un poco especial.

2.2.1. Código Java de Fragment 1


```
public class FragmentOne extends Fragment {
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_1,
                                container, false);
    }
}
```

Inflater pasado como parámetro

Ejecutamos el método *inflate*

Layout

2.2.2. Código xml de Fragment 1


```
<LinearLayout
      xmlns:android="http://sc....."
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation=”vertical”
     android:background=”#00ffff”
   <TextView
          android:id="@+id/textView1"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_weight="1"
          android:text="Fragmento 1"
          android:textStyle="bold"/>
</LinearLayout>
```

2.2.3. Código Java de Fragment 2


```
public class FragmentTwo extends Fragment {
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_2,
                                container, false);
    }
}
```

2.2.4. Código xml de Fragment 2

```
<LinearLayout
      xmlns:android="http://sc....."
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation=”vertical”
     android:background=”#ffff00”
   <TextView
          android:id="@+id/textView1"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_weight="1"
          android:text="Fragmento 2"
          android:textStyle="bold"/>
</LinearLayout>
```

2.4. Cargar el Fragment

- Heredamos de *AppCompactActivity*.
- En el fichero Layout definimos la etiqueta fragment.
  - Le indicamos un id para luego capturarlo en Java.
  - Le podemos indicar el fragmento por defecto que queremos cargar en el arranque de la Activity.

2.4.1. Código Java de MainActivity

```
public class MainActivity extends AppCompactActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    Public void selectFrag(View view) {
        Fragment fr;
        If  (view == findViewById (R.id.button1))
            fr = new FragmentOne();
         else
            fr = new FragmentTwo();
        FragmenManager fm = getFragmentManager();
        FragmenTransaction fragmentTransaction =
            fm.beginTransaction();
        FragmentTransaction.replace (R.id.fragment_place, fr);
        FragmentTransaction.commit();
    }
}
```

Elige el fragment

Manejador de fragmentos

Iniciamos transacción

Reemplazamos el fragmento indicado

Realizo los cambios

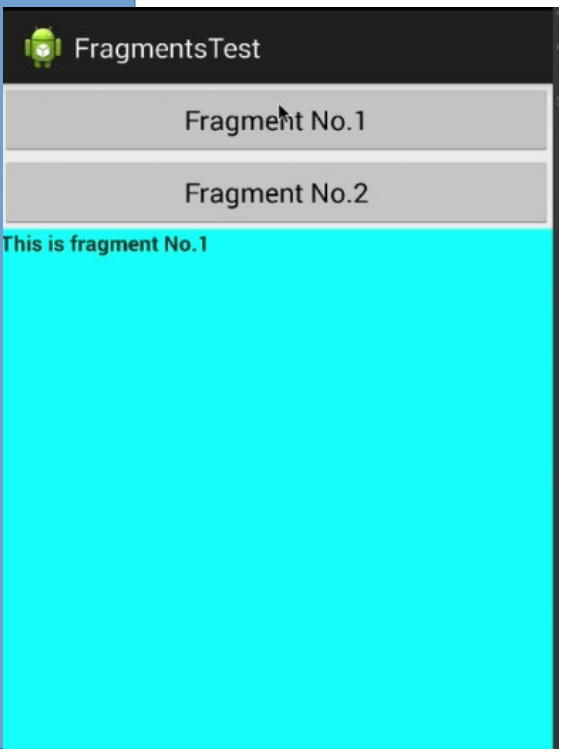
2.4.2. Código xml de MainActivity

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation=”vertical”
  <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Fragmento 1"
        android:onClick="SelectFrag"/>
<Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Fragmento 2"
        android:onClick="SelectFrag"/>
```

2.4.2. Código xml de MainActivity

```
<fragment
    Class = “com.manwest...”
    android:id="@+id/button1"
    android:layout_width="match_parent"
     android:layout_height="match_parent" />
</LinearLayout>
```

Clase que define a ese fragmento



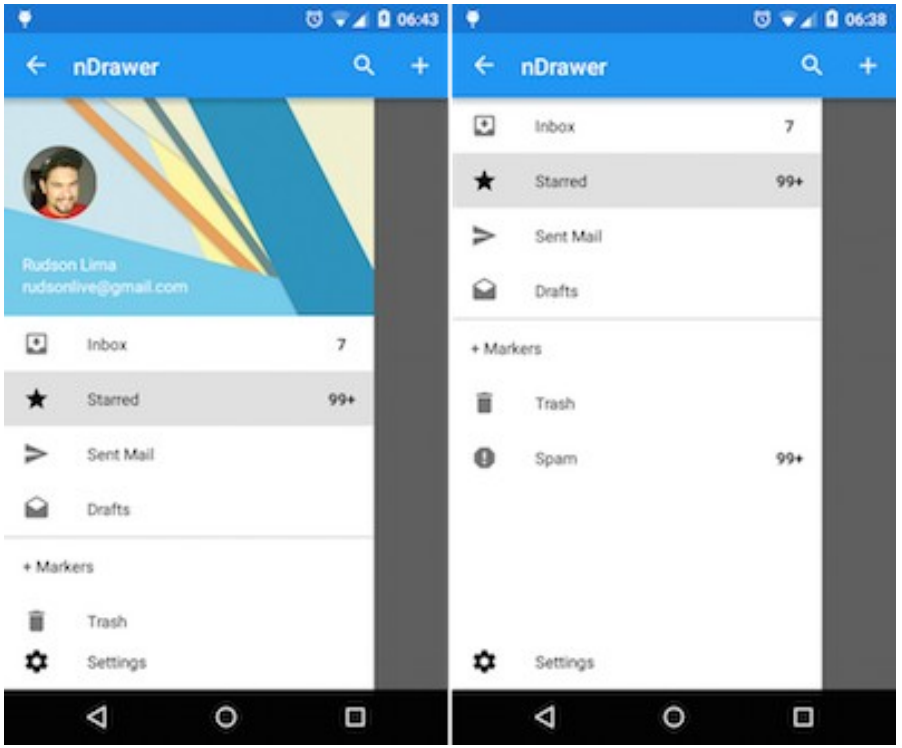
2.4.3. Fichero *Manifest*


```
<activity
  android:name=".FragmentOne"
  android:label="@string/app_name" >
   <intent-filter>
       <action android:name="android.intent.action.FragmentOne" />
       <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Indicamos los fragments.

Tantos como tengamos

3. Navegation View



3.1. Diseño

- Para añadir el Navigation Drawer a una actividad el layout XML sea del tipo:
  - *<android.support.v4.widget.DrawerLayout>*
- Dentro de este elemento colocaremos únicamente 2 componentes principales (en el orden indicado):
  - El layout real de la actividad.
  - El layout del menú lateral.

3.1. XML de Navigation (I)

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

    <!-- Layout real de la actividad -->
    <include layout="@layout/content_layout" />
```

Otorga efecto de aparición debajo del status bar

Xml real de la actividad que incluirá el Navigation

3.1. XML de Navigation (II)


```
<!-- Layout del menú lateral (Navigation View) -->
<android.support.design.widget.NavigationView
    android:id="@+id/navview"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:layout_gravity="start"
    app:headerLayout="@layout/header_navview"
    app:menu="@menu/menu_navview" />

</android.support.v4.widget.DrawerLayout>
```

"start" aparece por la izda y "end" por la dcha

Diseño de la cabecera del Navigation

Recurso del menu para cada opción

3.2. XML de Cabecera

```
<FrameLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <ImageView
     android:layout_width="match_parent"
     android:layout_height="200dp"
     android:src="@drawable/navheader"
     android:scaleType="centerCrop" />
  <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/usuario"
     android:textAppearance="@style/TextAppearance.AppCompat.Large.Inverse"
     android:textStyle="bold"
     android:layout_gravity="bottom"
     android:layout_marginBottom="10dp"
     android:layout_marginLeft="10dp" />
</FrameLayout>
```

Layout tipo Frame!!!!

Propiedades de la imagen

texto que queramos incrustar

3.3. XML de menu (I)


```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/menu_seccion_1"
        android:icon="@drawable/ic_menu"
        android:title="@string/seccion_1"/>
    <item
        android:id="@+id/menu_seccion_2"
        android:icon="@drawable/ic_menu"
        android:title="@string/seccion_2"/>
    <item
        android:id="@+id/menu_seccion_3"
        android:icon="@drawable/ic_menu"
        android:title="@string/seccion_3"/>
</group>
```

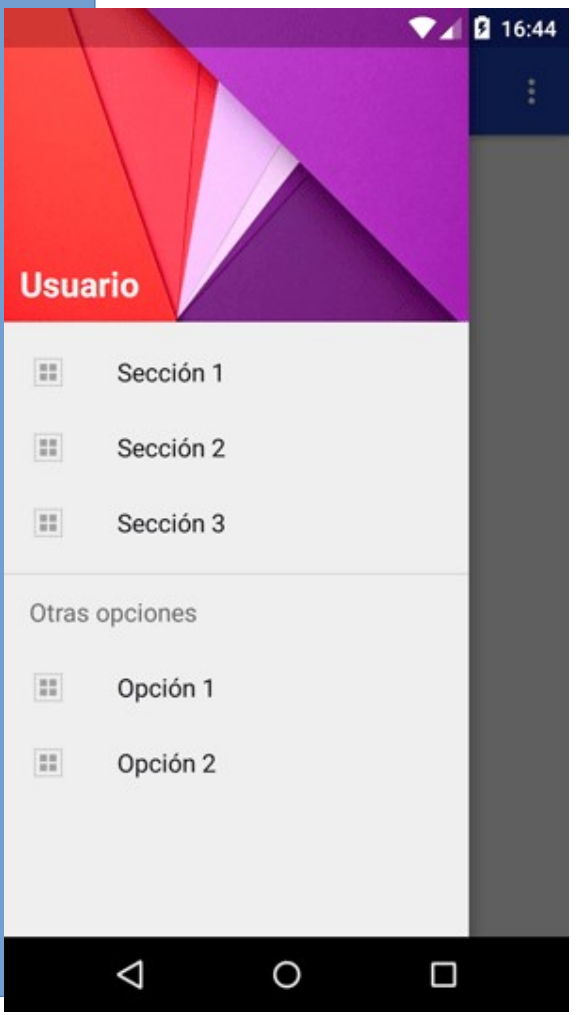
Sección 1 del menú

Cada una de los items

3.3. XML de menu (II)

```
<item
    android:id="@+id/navigation_subheader"
    android:title="@string/otras_opciones">
    <menu>
        <item
            android:id="@+id/menu_opcion_1"
            android:icon="@drawable/ic_menu"
            android:title="@string/opcion_1"/>
        <item
            android:id="@+id/menu_opcion_2"
            android:icon="@drawable/ic_menu"
            android:title="@string/opcion_2"/>
    </menu>
 </item>
</menu>
```

Sección 2 del menú



3.4. XML principal (I)

- Es el contenido en onCreate de la Activity principal.
  - Es el *content_layout* definido en *include* anteriormente
- Cada Sección será un fragment diferente

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:clickable="true"
   tools:context=".MainActivity"
   tools:showIn="@layout/activity_main">
```

3.4. XML principal (II)

```
<!-- Toolbar -->
   <android.support.v7.widget.Toolbar
      xmlns:app="http://schemas.android.com/apk/res-auto"
      android:id="@+id/appbar"
      android:layout_height="?attr/actionBarSize"
      android:layout_width="match_parent"
      android:minHeight="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      android:elevation="4dp"
      android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
      app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
    <FrameLayout
      android:id="@+id/content_frame"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

Diferentes propiedades de Toolbar

Resto de la interfaz de usuario

3.5. XML de cada Fragment

- Cada opción tendrá su propio Fragmen creado, por tanto tantos Fragment como opciones haya

```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".navigationdrawer.Fragment1">

   <TextView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:text="@string/fragment1" />

</FrameLayout>
```

3.6. Java de cada Fragment

- Cada Fragment con su Java definido.

```
public class Fragment1 extends Fragment {
    public Fragment1() {
        // Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_fragment1, container, false);
    }
}
```

3.7. Java de Activity Main (I)

- En onCreate para cada elemento que pueda seleccionar el usuario

```
drawerLayout = (DrawerLayout)findViewById(R.id.drawer_layout);
navView = (NavigationView)findViewById(R.id.navview);
 navView.setNavigationItemSelectedListener(
   new NavigationView.OnNavigationItemSelectedListener() {
      @Override
      public boolean onNavigationItemSelected(MenuItem menuItem) {
          boolean fragmentTransaction = false;
         Fragment fragment = null;
          switch (menuItem.getItemId()) {
            case R.id.menu_seccion_1:
                fragment = new Fragment1();
                fragmentTransaction = true;
                break;
           // Cada opción con su case correspondiente
         }
```

Declaración del componente

Cada una de las opciones que establecerán el fragment y la transacción a él

3.7. Java de Activity Main (II)

```
        if(fragmentTransaction) {
            getSupportFragmentManager().beginTransaction()
                .replace(R.id.content_frame, fragment)
                .commit();

            menuItem.setChecked(true);
            getSupportActionBar().setTitle(menuItem.getTitle());
        }

        drawerLayout.closeDrawers();

        return true;
    }
});
```

Transición al nuevo fragment seleccionado

Muestra la opción pulsada con true

Actualiza el título de la ActionBar

Cierra el menú

3.7. Java de Activity Main (III)

- Para crear nuestros iconos en diferentes resoluciones:
  https://romannurik.github.io/AndroidAssetStudio/index.html
- Añadimos el icono en *onCreate*

```
appbar = (Toolbar)findViewById(R.id.appbar);
setSupportActionBar(appbar);

getSupportActionBar().setHomeAsUpIndicator(R.drawable.ic_nav_menu);
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Para que sea un icono nuestro y no las tres barras (si se omite)

3.7. Java de Activity Main (IV)

- Deberemos capturar para que se abra nuestro Navigation

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch(item.getItemId()) {
        case android.R.id.home:
            drawerLayout.openDrawer(GravityCompat.START);
            return true;
        //...
    }

    return super.onOptionsItemSelected(item);
}
```

En caso de pulsarlo abre nuestro Navigation

4. Recursos Android

4.1. Recursos disponibles

- *drawable*: archivos de imagen
- *layout*: ficheros de diseño de pantallas
- *menu*: ficheros de diseño de menús
- *values*: define los literales, dimensiones y estilos
- *animator*: ficheros *XML* de definición de animaciones de propiedad
- *anim*: como la anterior pero para animaciones más complejas
- *color*: ficheros *XML* de definición de colores que luego pueden ser aplicados a propiedades o estilos
- *raw*: para almacenar archivos con información complementaria, accesible desde *Resources.openRawResource()*
- *XML*: permite almacenar información no clasificada en formato *XML*, accesible desde *Resources.getXML()*

4.2. Definición y ejemplos

- Tenemos dos opciones principales:
  - Clase *R*: para el acceso desde el código Java, es una clase autogenerada con el contenido del directorio */res*.
  - Referencias @: accesible desde las propiedades de un objeto en *XML*.
- Clase R:
  - *textview.setText(R.string.literal);*
- Referencia *XML*:
  - *<color name="opaque_red">#f00</color>*
  - *android:textColor="@color/opaque_red"*
- Usos de recursos de Sistema:
  - *android:textColor="?android:textColorSecondary"* (color dado por el sistema)
  - *android.R.layout.simple_list_item_1*

5. Localización

5.1. Definición

- Debemos pensar en que nuestra aplicación debe poder ser utilizable por otras personas que no hablan nuestro idioma
- Deberemos identificar aquellos literales que deban ser traducibles
- Es posible que sea necesario identificar también aquellas imágenes que deban ser traducidas, por ejemplo banderas.
  - http://developer.android.com/guide/topics/resources/index.html
- Tablas de Códigos de Traducción
  - Basados en el estándar ISO-639-1 e ISO-3166-1
  - es, en, es-ES, en-US

5.2. Recursos

- */res/values Idioma por defecto*
- */res/values-es Idioma en español*
- */res/values-en Idioma en inglés*
- */res/layout layout por defecto*
- */res/layout-es layout en español*
- */res/layout-en layout en inglés*
- *igual para drawable, anim, xml, raw*

5.3. Buenas prácticas

- Diseñar las pantallas de una manera flexible
- Soportar RTL (Right To Left).
  - Por ejemplo para árabes o hebreos.
- Utilizar layouts alternativos si es necesario.
- Usar fechas, horas, números y cantidades basadas en el sistema.
- Utilizar siempre que sea posible literales traducibles.
- http://developer.android.com/distribute/tools/localization-checklist.html

6. Tipos de recursos

6.1. Drawable

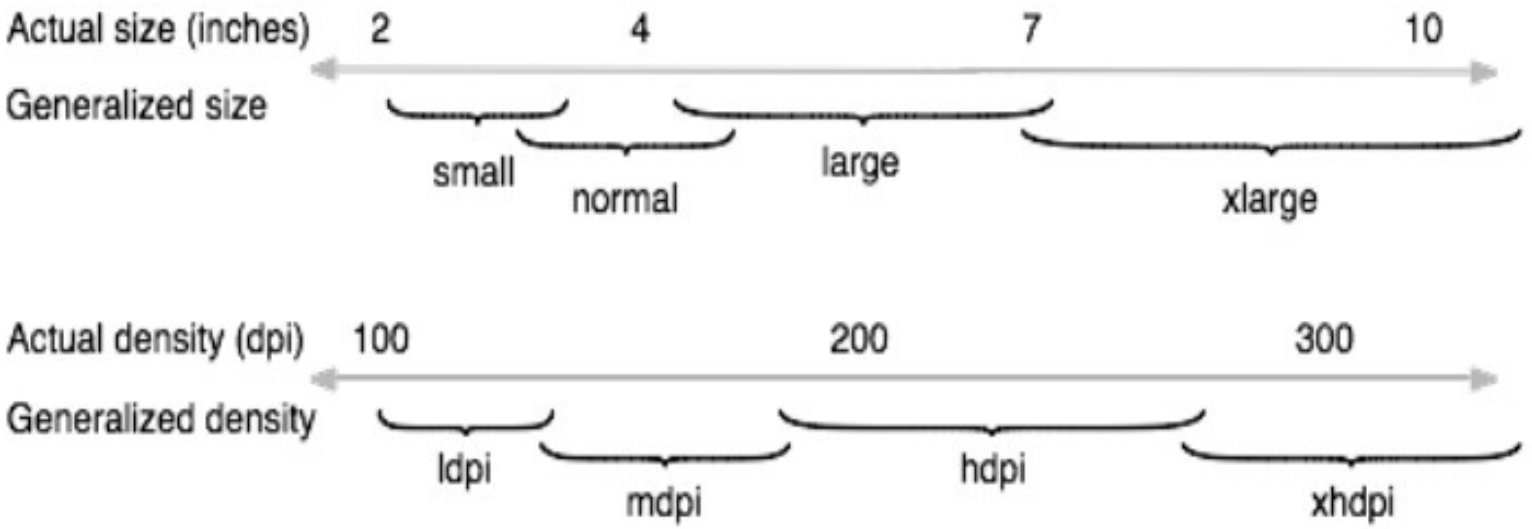
- -*ldpi*: baja densidad,; aproximadamente 120dpi
- -*mdpi*: media densidad(HVGA); aproximadamente 160dpi
- -*hdpi*: alta densidad; aproximadamente 240dpi
- -*xhdpi*: extra alta densidad; aproximadamente 320dpi.
  - Añadido en el API Level 8
- -*xxhdpi*: 2x extra alta densidad; aproximadamente 480dpi.
  - Añadido en el API Level 16
- -*xxxhdpi*: 3x alta densidad(sólo para el icono del lanzador, ver información sobre el diseño en pantallas múltiples); aproximadamente 640dpi.
  - Añadido en el API Level 18
- directorio simplemente "*drawable*": común a todos los tipos de pantallas

6.2. Layout (I)

- */layout:* directorio por defecto
- *-ldrtl:* para idiomas de derecha a izquierda
- *-sw320dp:* para dispositivos con resolución de al menos 320dp tanto en alto como en ancho.
- *-w720dp:* para dispositivos con una resolución de al menos 720dp de ancho.
- *-h720dp:* para dispositivos con una resolución de al menos 720dp de alto
- *-land:* para visualización en vertical
- *-port:* para visualización en horizontal
- *-long:* para pantallas alargadas
- *-nolong:* para pantallas más bien cuadradas

6.2. Layout (II)

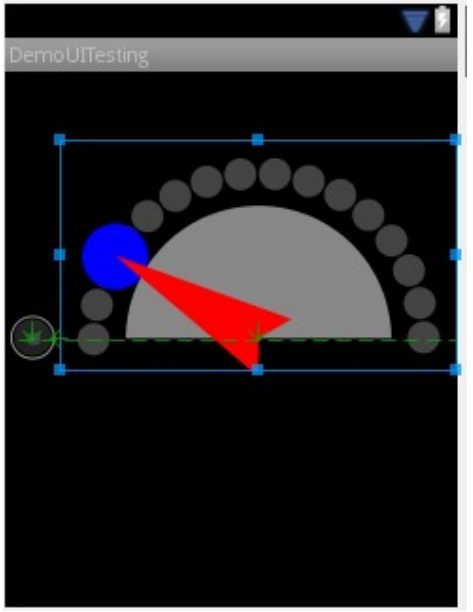
- Varios directorios de */layout*:
  - *-small, -normal, large, xlarge*



6.2. Layout (III)

- ejemplos de layouts:
  - *res/layout/my_layout.xml*
  - *res/layout-large/my_layout.xml*
  - *res/layout-xlarge/my_layout.xml*
  - *res/layout-land/my_layout.xml*
  - *res/layout-port/my_layout.xml*
  - *res/layout-xlarge-land/my_layout.xml*

7. Custom View



6.1. Introducción

- Crear nuestros controles personalizados:
  - Extendiendo funcionalidad de existente.
  - Combinando varios controles formando uno complejo.
  - Diseñándolo desde cero.

6.2. Extender funcionalidad (I)


```
public class ExampleCustom extends EditText {
    private Paint p1;
    private Paint p2;
    private float escala;
    public ExtendedEditText(Context context, AttributeSet attrs, int defStyle){
        super(context, attrs, defStyle);
        inicializacion();
    }
    public ExtendedEditText(Context context, AttributeSet attrs) {
        super(context, attrs);
        inicializacion();
    }
    public ExtendedEditText(Context context) {
        super(context);
        inicializacion();
    }
```

Clase padre del componente

Constructores

6.2. Extender funcionalidad (II)

Clase padre del componente

```
private void inicializacion()
    {
        p1 = new Paint(Paint.ANTI_ALIAS_FLAG);
        p1.setColor(Color.BLACK);
        p1.setStyle(Style.FILL);

        p2 = new Paint(Paint.ANTI_ALIAS_FLAG);
        p2.setColor(Color.WHITE);
        p2.setTextSize(20);

        escala = getResources().getDisplayMetrics().density;
    }
```

Establece características que se darán a objetos a insertar (elipse y texto)

6.2. Extender funcionalidad (III)


```
@Override
public void onDraw(Canvas canvas) {
  //Llamamos al método de la clase base (EditText)
  super.onDraw(canvas);

  //Dibujamos el fondo negro del contador
  canvas.drawRect(this.getWidth()-30*escala,
        5*escala,
        this.getWidth()-5*escala,
        20*escala, p1) ;

  //Dibujamos el número de caracteres sobre el contador
  canvas.drawText("" + this.getText().toString().length(),
        this.getWidth()-28*escala,
        17*escala, p2);
}
```

Reescribimos el evento *onDraw* cada vez que hay que redibujar el control en pantalla

Canvas proporciona los métodos para dibujar elementos (líneas, rectángulos, elipses…). Dos objetos fondo (drawRect) y texto (drawText)

6.2. Extender funcionalidad (IV)

Clase padre del componente

```
<RelativeLayout ....>

    <nombrePaquete.ExampleCustom
     android:layout_width="match_parent"
     android:layout_height="wrap_content" />

</RelativeLayout ....>
```

Nombre del paquete y clase creada