Temari

1) Introducción
2) Framework Android
3) Proyectos Android y Android SDK
4) Activity
5) Fragments, Views y ListViews
6) Intents
7) Layouts y Custom Views
8) Resources y Themes
9) Dialogs, Menus y WebView
10) Persisténcia de datos
11) Tareas en Background e internet
12) SQLite y content providers
13) Notificaciones

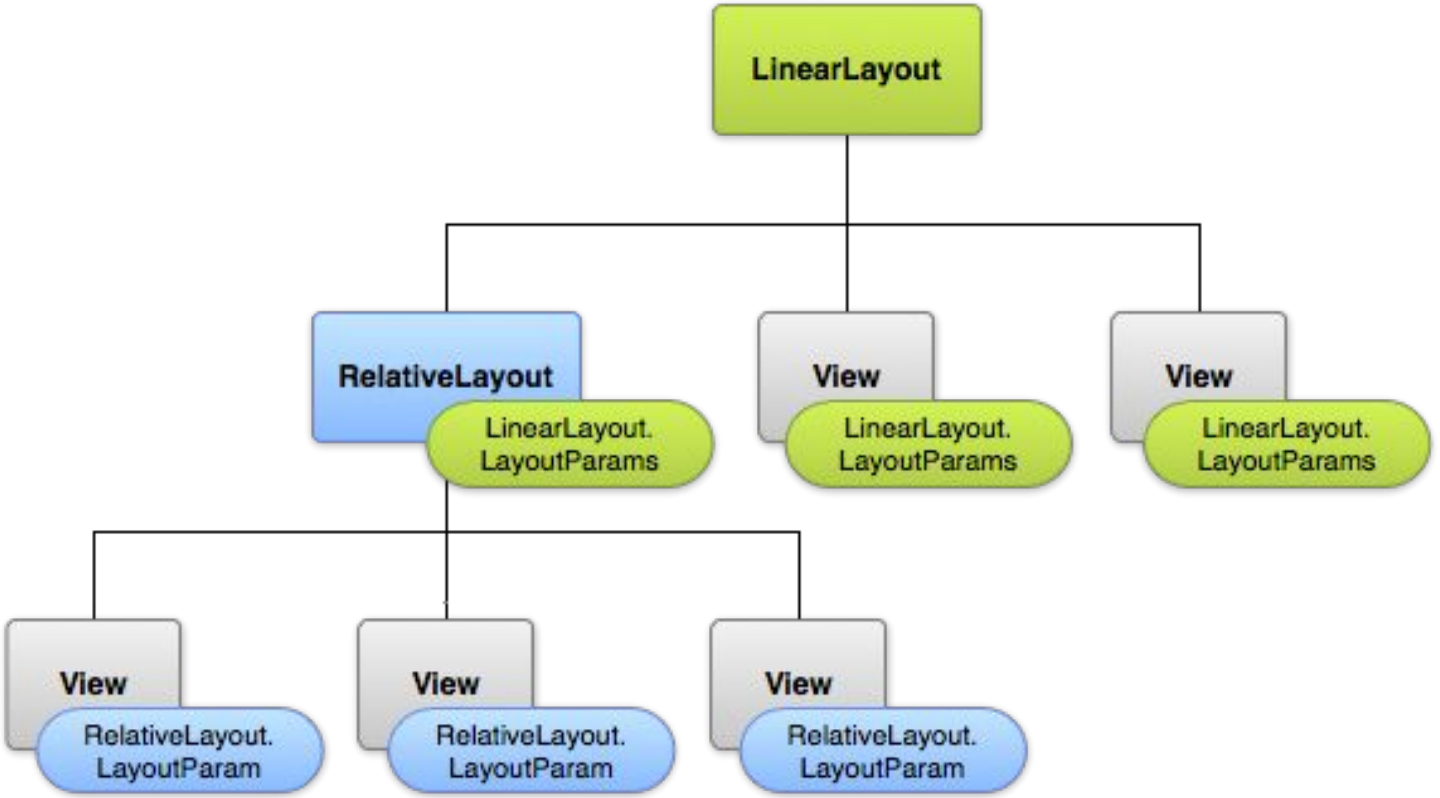
7 – Layouts y Custom Views

- Layouts disponibles
- Propiedades de los Layouts
- Views propias
- Modificar Views existentes

Layouts

Estructuren un grup de Views en una porció de la UI.

Diferents layouts per estructures diferents



Layouts

- **LinearLayout**
- **RelativeLayout**
- **TableLayout**
- **FrameLayout**
- **GridLayout**
- **DrawerLayout**
- **CoordinatorLayout**
- **...**

Todas subclass de **ViewGroup**

Layout Properties

Cualquier ViewGroup contiene una *nested class* del tipo **ViewGroup.LayoutParams**, que define como se distribuyen las Views que contiene

- Es **obligatorio** definir un layout_**width** i un layout_**height,** que pueden ser**:**
  - Un valor exacto, en **dpi**
  - **match_parent**
  - **wrap_content**
- Otras propiedades:
  - layout_margin[Botton,Top,Right,Left]
  - layout_gravity → cómo se posicionan las Views que contiene
  - layout_weight → cómo se distribuye el espacio sobrante del layout
  - layout_x i layout_y: las coordenadas del layout

Inflar con un layout

- **En la Activity** → setContentView(int resourceId)
- **En un Fragment** → inflater.inflate(int resourceId, ViewGroup container, boolean)

Custom Views

A parte de las Views que por defecto proporciona Android, si tenemos una necessidad especial podemos hacer una **Custom.** Hay 3 vies:

- **Modificar una View existente**: Las Views existents tienen parámetros para configurar aspectos: el color, la medida, la dirección del texto...
- **Combinar várias Views en una neuva**: combinar varias Views en una nueva que las agrupa todas, y usar ésta como una sola View
- **Implementar una View completamente nueva**: Crear una View **extendiendo** la classe original: android.view.View

Modificar una View existente

- **extends** de la View que se queire modificar

- Crear constructores (llamar al de la superclass!)
  - Pasar por lo menos el **Context** i el **AttributeSet** → de esta forma aparece en el layout editor

- Sobreescribir métodos:
  - **Init():** Inicializar objetos Paint que usarem
  - **onDraw(Canvas canvas)**: Utilitzando objetos Paint, pintar sobre el canvas formas, texto, imágenes

Canvas: **Qué forma se pinta**

Paint: **Cómo se pinta**

Compound Views (I)

- **Definir atributos opcionales** en un fichero de recursos res/values/attrs.xml → Seran los atributos para configurar la View en los XML de layout en los que pueda aparecer

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Options">
        <attr name="titleText" format="string" localization="suggested" />
        <attr name="valueColor" format="color" />
    </declare-styleable>

</resources>
```


Compound Views (II)

- Crear un XML que combina las diferentes Views en un elemento *<merge>*
- Crear la classe correspondiente → extends un Layout
- En código, sobreescribir el constructor que al que se passa **Context** i **AttributeSet**
    - Obtener valores de las configuraciones en un **TypedArray** del Context

```
TypedArray a = context.obtainStyledAttributes(attrs,
    R.styleable.ColorOptionsView, 0, 0);
String titleText = a.getString(R.styleable.ColorOptionsView_titleText);
int valueColor = a.getColor(R.styleable.ColorOptionsView_valueColor,
    android.R.color.holo_blue_light);
a.recycle();
```


Compound Views (III)

- Inflar la clase de la Custom View con el XML que combinava las Views

```
LayoutInflater inflater = (LayoutInflater) context
    .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
inflater.inflate(R.layout.view_color_options, this, true);

TextView title = (TextView) getChildAt(0);
title.setText(titleText);
```

- Configurar las Views como se necesite en el código

View nueva (I)

- Crear una nueva class que extienda **android.view.View**
- Crear nuevos recursos **declare-styleable** en el fichero */res/values/attrs.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Options">
        <attr name="titleText" format="string" localization="suggested" />
        <attr name="valueColor" format="color" />
    </declare-styleable>

</resources>
```

- Declarar variables de tipo Paint que necesitaremos para pintar los diferentes elementos

View nueva (II)

- Definir un constructor: **public MyView(Context ctx, AttributeSet attrs)**
    - Obtener los atributos del XML del **TypedArray** (los definidos en el ***attrs.xml)***

```
public PieChart(Context context, AttributeSet attrs) {
   super(context, attrs);
   TypedArray a = context.getTheme().obtainStyledAttributes(
        attrs,
        R.styleable.PieChart,
        0, 0);

   try {
       mShowText = a.getBoolean(R.styleable.PieChart_showText, false);
       mTextPos = a.getInteger(R.styleable.PieChart_labelPosition, 0);
   } finally {
       a.recycle();
   }
}
```


View nueva (III)

Inicializar los objetos Paint en un método **init**()

Sobreescriubir el **onDraw**()

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Draw the shadow
    canvas.drawOval(
            mShadowBounds,
            mShadowPaint
    );

    // Draw the label text
    canvas.drawText(mData.get(mCurrentItem).mLabel, mTextX, mTextY, mTextPaint);
```


View nueva (IV)

- Implementar getters y setters
    - Después de cambiar atributos que modifiquen la visualización, hay que llamar a **invalidate() i requestLayout**()