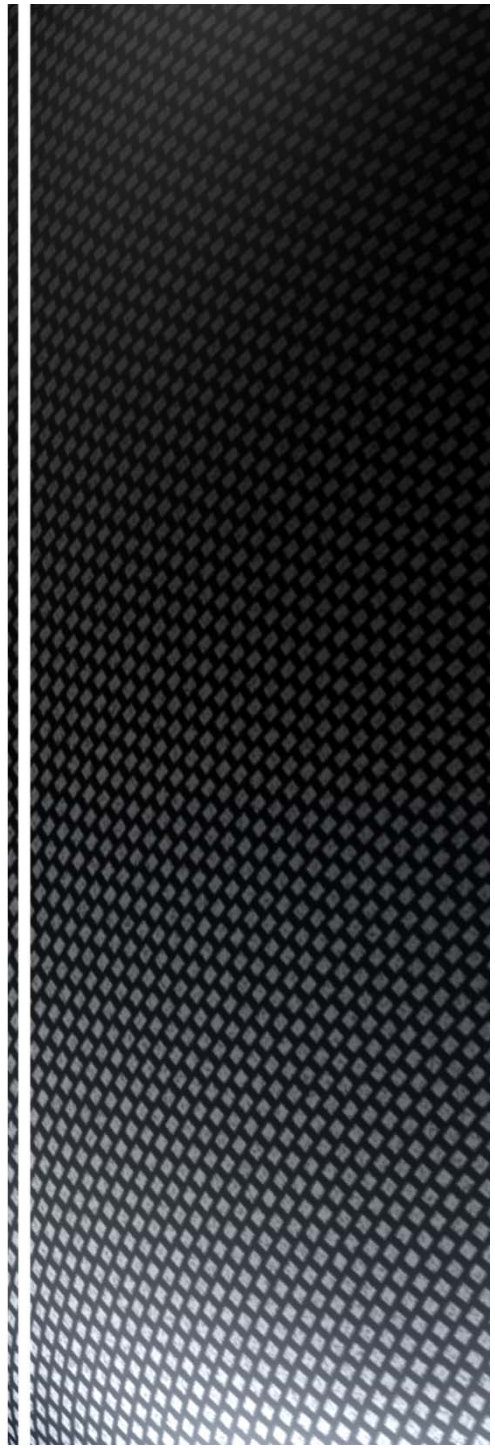
Gráficos



Gráficos

- Android nos proporciona a través de su API gráfica una potente colección de funciones que cubren prácticamente cualquier necesidad gráfica de una aplicación.
  - Manipulación de imágenes,
  - Gráficos vectoriales,
  - Animaciones,
  - Trabajo con texto o gráficos en 3D

Clases APIs Gráficas

Para 2D

- Canvas: representa una superficie donde podemos dibujar.
- Paint: permite definir color, grosor, estilo de un gráfico vectorial
- Path: permite definir un trazado a partir de segmentos de rectas y curvas
- Drawable: Abstracción que representa “algo que puede ser dibujado”
  - BitmapDrawable -> basado en mapas de bits
  - ShapeDrawable -> a partir de primitivas vectoriales
  - AnimationDrawable -> frame a frame a partir de objetos drawable.

Para 3D

- OpenGL: podemos usar las API estandar de OpenGL.
- Renderscript: Nueva API para renderizado a bajo nivel (a partir del SDK 3.0)
  - Programadas en lenguaje C99 y luego compilado a código nativo del procesador.

Canvas

- Para dibujar en un Canvas necesitaremos un pincel (Paint) donde definiremos el color, grosor de trazo, transparencia,…
- También podremos definir una matriz de 3x3 (Matrix) que nos permitirá transformar coordenadas aplicando una translación, escala o rotación.
- Otra opción consiste en definir un área conocida como Clip, de forma que los métodos de dibujo afecten solo a esta área.

  consultar la documentación del SDK para una información más detallada de todas las funciones posibles y parametros.

- Para dibujar figuras geométricas

```
drawCircle(float cx, float cy, float radio, Paint pincel)
drawOval(RectF ovalo, Paint pincel)
drawRect(RectF rect, Paint pincel)
drawPoint(float x, float y, Paint pincel)
drawPoints(float[] pts, Paint pincel)
```

- Para dibujar líneas y arcos

```
drawLine(float iniX, float iniY, float finX, float finY,
Paint pincel)
drawLines(float[] puntos, Paint pincel)
drawArc(RectF ovalo, float
iniAnglulo, float anglulo, boolean usarCentro,Paint pincel)
drawPath(Path trazo, Paint pincel)
```

Canvas

- Para dibujar Texto

```
drawText(String texto, float x, float y, Paint pincel)
drawTextOnPath(String texto, Path trazo, float desplazamHor,
               float desplazamVert, Paint pincel)
drawPosText(String texto, float[] posicion, Paint pincel)
```

- Para rellenar todo el Canvas (a no ser que se haya definido un Clip)

```
drawColor(int color)
drawARGB(int alfa, int rojo, int verde, int azul)
drawPaint(Paint pincel)
```

- Para dibujar Imágenes

```
drawBitmap(Bitmap bitmap, Matrix matriz, Paint pincel)
```

- Si definimos un Clip, solo se dibujará en el área indicada

```
boolean clipRect(RectF rectangulo)
boolean clipRegion(Region region)
boolean clipPath(Path trazo)
```

Canvas

- Definir una matriz de transformación (Matrix) nos permitirá transformar coordenadas aplicando una translación, escala o rotación

```
setMatrix(Matrix matriz)
Matrix getMatrix()
concat(Matrix matriz)
translate(float despazX, float despazY)
scale(float escalaX, float escalaY)
rotate(float grados, float centroX, float centroY)
skew(float despazX, float despazY)
```

- Para saber el tamaño del Canvas

```
int getHeight()
int getWidth()
```

Color

- Android representa los colores con enteros de 32 bit, divididos en 4 campos de 8 bits: alfa, rojo, verde y azul (ARGB). Cada componente tomará 256 valores diferentes.
- Los componentes rojo, verde y azul definen el color, mientras que el componente alfa define el grado de transparencia. Un valor 255 significa opaco.
- Para definir un color tenemos las siguientes opciones:

```
setColor(int color)
        int color;
        color = Color.BLUE;                  //Azul opaco
        color = Color.argb(127, 0, 255, 0);  //Verde transparente
        color = 0x7F00FF00;                  //Verde transparente
        color = getResources().getColor(R.color.color_circulo);
setAlpha(int alfa)
```

Separación entre programación y diseño:
NO: definir los colores en código
SI : usar fichero de Recursos res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="color_circulo">#7fffff00</color>
</resources>
```

Paint

La mayoría de los métodos de la clase Canvas utilizan un parámetro de tipo Paint.

Esta clase nos permite definir el color, estilo o grosor del trazado de un gráfico vectorial.

Definición del Trazo

`setStrokeWidth(float grosor)`

grosor del trazo

`setStyle(Paint.Style estilo)`

cómo se interpretan las primitivas gráficas, FILL, FILL_AND_STROKE, STROKE

`setShadowLayer(float radio, float dx, float dy, int color)`

realiza un segundo trazo a modo de sombra

Definición del Texto

`setTextAlign(Paint.Align justif)`

tipo de justificación: CENTER, LEFT, RIGHT

`setTextSize(float size)`

tamaño del texto

`setTypeface(Typeface fuente)`

tipo de fuente: MONOSPACE, SANS_SERIF, SERIF , fuentes adicionales se instalan como recursos

desde la clase Typeface podemos definir negrita/italica

`setTextScaleX(float escalaX)`

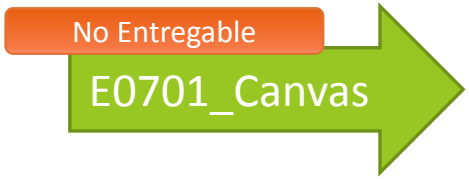
Factor de escalado horizontal, por defecto 0

`setTextSkewX(float inclinationX)`

factor de inclinación, por defecto 0

`setUnderlineText(boolean subrayado)`

texto subrayado



Path

- La clase Path permite definir un trazado a partir de segmentos de línea y curvas.
- Tras definirlo puede ser dibujado con canvas.drawPath(Path, Paint).
- Un Path también puede ser utilizado para dibujar un texto sobre el trazado marcado.

Animar a lo largo de un path

La clase PathMeasure oferta una serie de métodos para poder gestionar medidas y obtener posiciones de un path.

getLenght()

Devuelve la longitud del path

getPosTan(Float distance, float[] pos, float[] tan)

Dada una distancia desde el inicio del path devuelve en los vectores pos y tan la posición en (x,y) en el canvas y la tangente de esa distancia al origen del path.

La idea para animar un objeto a lo largo de un path es:

- Dividir el path en segmentos por los que iterar
- Calcular la posicion de cada segmento
- Dibujar el objeto en esa posicion
- Incrementar el indice del segmento
- Pausar el UI Thread para temporizar la animación.
- Gestionar la reentrada en onDraw()

No Entregable

E0702_Path

El método onDraw de la View es llamado cada vez que ésta necesita redibujarse.

Para forzar la llamada al método deberemos invalidar la vista usando el método invalidate()

Path

Animar a lo largo de un path

El siguiente código es un ejemplo de iteración.

```
int ixSegmento = 0;
int numSegmentos = 100;
PathMeasure pm = new PathMeasure(trazo, false);
float fSegmentLen = pm.getLength() / numSegmentos;
float afPosition[] = {0f, 0f};

if (ixSegmento <= numSegmentos) {
    pm.getPosTan(fSegmentLen * ixSegmento, afPosition, null);
    canvas.drawCircle(afPosition[0],afPosition[1],10,pincel);
    ixSegmento++;
    invalidate();
    try {
        Thread.sleep(20);
        }
    catch (InterruptedException e) {
        e.printStackTrace();
        }
    }
else {
    ixSegmento = 0;
};
```

Drawable

Clase Drawable es una abstracción que representa “algo que puede ser dibujado”.

Esta clase se extiende para definir gran variedad de objetos gráficos más específicos.

Pueden ser definidos como recursos usando ficheros XML.

- BitmapDrawable

  Imagen basada en un fichero gráfico (PNG o JPG). Etiqueta XML <bitmap>.

- ShapeDrawable

  Permite realizar un gráfico a partir de primitivas vectoriales, como formas básicas (círculos, cuadrados,…) o trazados (Path).

  No puede ser definido mediante un fichero XML.

- LayerDrawable

  Contiene un array de Drawable que son visualizados según el orden del array.

  Mayor indice del array es el que se representa encima.

  Cada Drawable puede situarse en una posición determinada. Etiqueta XML <layer-list>

- StateListDrawable

  Similar al anterior pero ahora podemos usar una máscara de bits podemos seleccionar los objetos visibles. Etiqueta XML <selector>.

- GradientDrawable

  Degradado de color que puede ser usado en botones o fondos.

- TransitionDrawable

  Una extensión de LayerDrawables que permite un efecto de fundido entre la primera y la segunda capa. Para iniciar la transición hay que llamar a startTransition(int tiempo). Para visualizar la primera capa hay que llamar a resetTransition(). Etiqueta XML<transition>.

- AnimationDrawable

  Permite crear animaciones frame a frame a partir de una serie de objetosDrawable. Etiqueta XML <animation-list>

Drawable

Drawable proporciona una serie de mecanismos genéricos que permiten indicar como ha de ser dibujada.

`setBounds(x1,y1,x2,y2)`

Permite indicar el rectángulo donde ha de ser dibujado.

Todo Drawable debe respetar el tamaño solicitado por el cliente, es decir, ha de permitir el escalado. Podemos consultar el tamaño preferido de un Drawable mediante los métodos getIntrinsicHeight() y getIntrinsicWidth().

`getPadding(Rect)`

Nos proporciona información sobre los márgenes recomendados para representar contenidos.

Por ejemplo, un Drawable que intente ser un marco para un botón, debe devolver los márgenes correctos para localizar las etiquetas, u otros contenidos, en el interior del botón.

`setState(int[])`

Permite indicar al Drawable en qué estado ha de ser dibujado, por ejemplo "con foco", "seleccionado", etc.

Algunos Drawable cambiarán su representación según este estado.

`setLevel(int)`

permite implementar un controlador sencillo para indicar como se representará el Drawable.

Por ejemplo, un nivel puede ser interpretado como una batería de niveles o un nivel de progreso. Algunos Drawable modificarán la imagen basándose en el nivel.

Un Drawable puede realizar animaciones al ser llamado desde el cliente a su método Drawable.Callback.

Todo cliente debe implementar esta interfaz via setCallback(Drawable.Callback)

El sistema nos facilita realizar esta tarea de forma sencilla a través de setBackgroundDrawable(Drawable) e ImageView.

Bitmap Drawable

- La forma más sencilla de añadir gráficos a tu aplicación es incluirlos en la carpeta res/drawable del proyecto.
- El SDK de Android soporta los formatos PNG, JPG y GIF.
- El formato aconsejado es PNG, aunque si el tipo de gráfico así lo recomienda también puedes utilizar JPG. El formato GIF está desaconsejado.
- Cada gráfico de esta carpeta es asociado a un ID de recurso.
- Por ejemplo, para el fichero mi_imagen.jpg se creará el ID mi_imagen.
- Este ID te permitirá hacer referencia al gráfico desde el código o desde XML.

Gradient Drawable

- También podemos definir en XML otro tipo de Drawables como GradientDrawable. Por ejemplo, el siguiente fichero define un degradado desde el color blanco (FFFFFF) a azul (0000FF):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
<gradient
  android:startColor="#FFFFFF"
  android:endColor="#0000FF"
  android:angle="270" />
</shape>
```

- Se suelen usar como fondo de botones o de pantalla.
- El parámetro angle indica la dirección del degradado.
- Solo se permiten los ángulos 0, 90, 180 y 270.

No Entregable

E0703_BitmapDrawable

Transition Drawable

- Un TransitionDrawable es una extensión de LayerDrawables que permite un efecto de fundido entre la primera y la segunda capa.
- Para iniciar la transición hay que llamar a startTransition(int tiempo).
- Para volver a visualizar la primera capa hay que llamar a resetTransition().

No Entregable

E0704_TransitionDrawable

ShapeDrawable

- Permite crear gráficos dinámicamente mediante primitivas vectoriales.
- Esta clase permite dibujar gráficos a partir de **formas** básicas.
- Un ShapeDrawable es una extensión de Drawable, por lo tanto se puede utilizar todo lo que permite Drawable.

No Entregable

E0705_ShapeDrawable

AnimationDrawable

- Varios mecanismos para crear animaciones.
- Ventaja: las animaciones pueden ser creadas en ficheros XML.
- Ejemplo animación a partir de fotogramas. Para ello utilizaremos la clase AnimationDrawable.

No Entregable

E0706_AnimationDrawable

Utilizar Vistas Reutilizables

Hasta ahora hemos creado una clase "EjemploView" que desciende de View dentro de la clase que contiene la Activity.

Más interesante generarla fuera, en una clase independiente,para poder usarla en cualquier parte de nuestro proyecto.

Cosas a Tener En Cuenta:

Para crear una nueva vista, hay que:

- Extender la clase View
- Escribir un constructor
- Como mínimo sobreescribir métodos:
  - onSizeChanged()
  - onDraw().

```
public class MiVista extends View {
   public MiVista(Context context, AttributeSet attrs) {
       super(context, attrs);
       //Inicializa la vista
       //OjO: Aún no se conocen sus dimensiones
   }
   @Override protected void onSizeChanged(int ancho, int alto,
            int ancho_anterior, int alto_anterior){
            //Te informan del ancho y el alto
   }
   @Override protected void onDraw(Canvas canvas) {
       //Dibuja aquí la vista
   }
}
```

Constructor:

Tiene dos parámetros:

- Context permitirá acceder al contexto de aplicación, p.e para utilizar recursos de esta aplicación.
- AttributeSet, permitirá acceder a los atributos de esta vista, cuando sea creada desde XML.

El constructor es el lugar adecuado para crear todos los componentes de la vista
OJO, en este punto todavía no se conoce las dimensiones que tendrá.

Utilizar Vistas Reutilizables

Cosas a Tener En Cuenta:

Varias Pasadas:

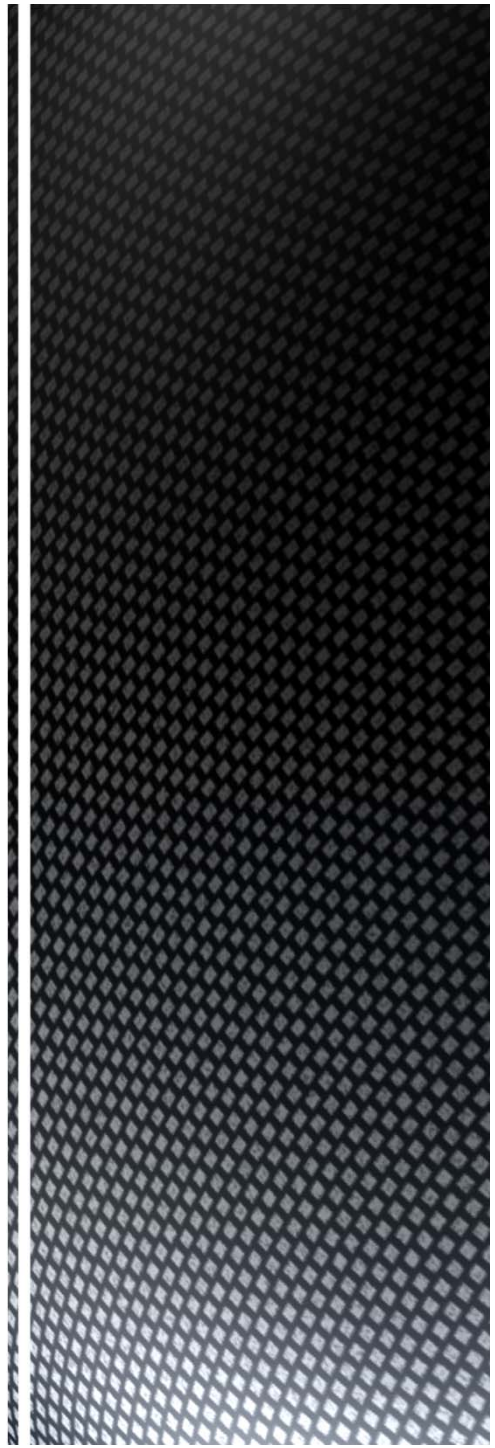
Android realiza un proceso de varias pasadas para determinar el ancho y alto de cada vista dentro de un Layout.

Cuando finalmente ha establecido las dimensiones de una vista llamará a su método onSizeChanged(), indicando como parámetros el ancho y alto asignado.

En caso de tratarse de un reajuste de tamaños, por ejemplo una de las vistas del Layout desaparece y el resto tienen que ocupar su espacio, se nos pasará el ancho y alto anterior.

Si es la primera vez que se llama al método estos parámetros valdrán 0.

Animaciones Tween



Transformaciones Tween

Una “animación tween” puede realizar series de transformaciones simples en el contenido de un View.

- posición
- tamaño
- rotación
- transparencia

La secuencia de órdenes que define la “animación tween” puede estar escrita mediante xml o código.

Cada intrucción (xml tag) define una transformación.

Cada instrucción define cuando ocurrirá y cuando tiempo tardará en completarse.

Las transformaciones pueden ser secuenciales o simultáneas.

Existen parámetros comunes a toda transformación y parámetros específicos para cada transformación.

El fichero XML que define a la animación debe pertenecer al directorio res/anim/ en tu proyecto Android.

El archivo debe tener solo un único elemento raíz: este debe ser uno de los siguientes:

- <translate>
- <rotate>
- <scale>
- <alpha>
- <set> que puede contener grupos de estos elementos (además de otro <set>)

Por defecto, todas las instrucciones de animación ocurren a partir del instante inicial.

Si quieres que una animación comience más tarde debes especificar el atributo startOffset.

Transformaciones Tween

Para cargar en Java un objeto Animation desde el XML utilizaremos la clase AnimationUtils

```
Animation mi_animacion= AnimationUtils.loadAnimation(context,resource)
```

Para lanzar una animación sobre un objeto View se llama al método .startAnimation pasándole el objeto Animation cargado con el recurso xml

Elementos de animación y parámetros en

http://developer.android.com/guide/topics/resources/animation-resource.html#Tween

No Entregable

E0708_AnimacionTween

E0709_CitasCelebres