Desarrollo de Aplicaciones para Android

Sesión 3: Gráficos avanzados

Puntos a tratar

- Personalización de componentes
- Elementos *drawables*
- Creación de componentes propios
- Lienzo y pincel
- Primitivas geométricas
- Texto
- Imágenes
- Gráficos 3D y OpenGL

Gráficos en Android

- Dos formas de personalizar gráficos
  - Alto nivel
    - Definimos elementos *drawables*
      - En XML o programados
    - Los utilizamos para personalizar componentes
      - Por ejemplo en `ImageView` o `Button`
  - Bajo nivel
    - Definir componente propio
    - Subclase de `View`
    - Especificamos cómo *pintar* el componente
      - Sobrescribiendo el método `onDraw`

Elementos *drawables*

- Se pueden aplicar a componentes
- Se definen en XML o de forma programática
  - Tipos básicos
    - Formas geométricas
    - Gradientes
    - Imágenes
    - *Nine-patch*
  - Tipos combinados
    - Capas
    - Estados
    - Niveles
    - Transiciones
    - Inserción
    - Recorte
    - Escala
    - Animaciones

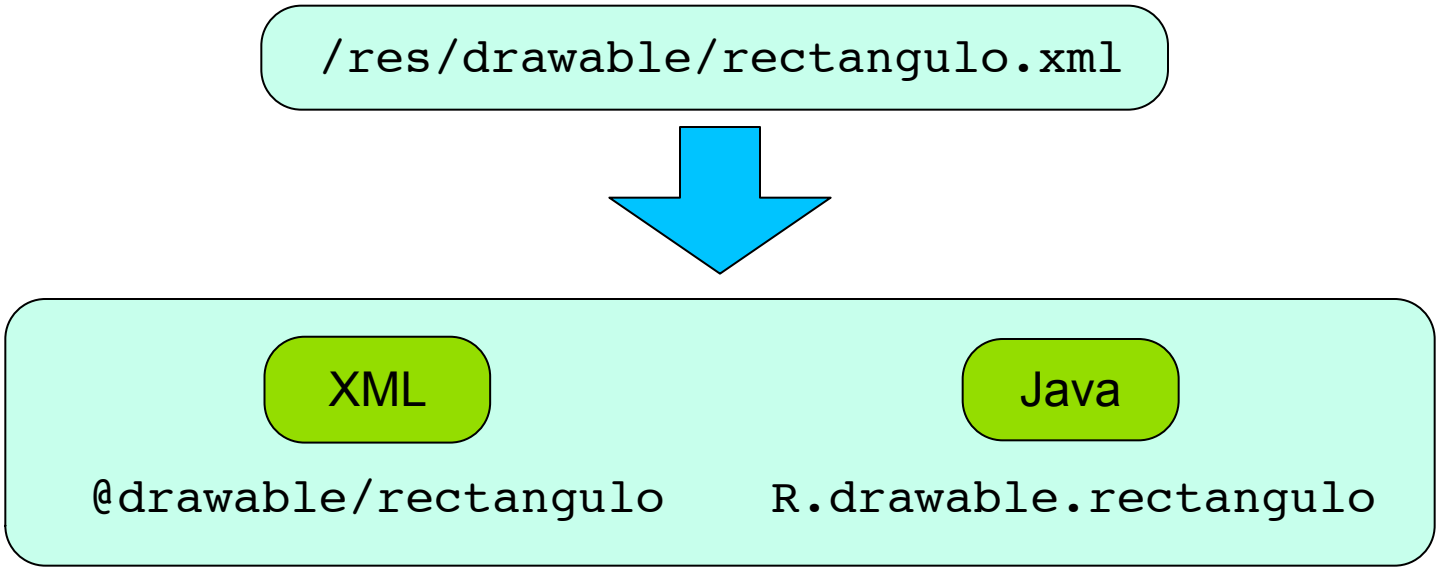
http://developer.android.com/guide/topics/resources/drawable-resource.html

Directorio de *drawables*

- Dentro de los directorios de recursos XML

`/res/drawable`

- Serán accesibles desde XML y Java



Variantes de los recursos

- Existen diferentes variantes de *drawables*
  - Densidad de pantalla

    ```
    drawable-ldpi drawable-mdpi drawable-hdpi drawable-xhdpi
    ```

  - Tamaño de pantalla

    ```
    drawable-small drawable-normal drawable-large drawable-xlarge
    ```

  - Orientación de la pantalla

    ```
    drawable-port drawable-land
    ```

- Se pueden aplicar a cualquier tipo de recurso

http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources

Drawables en XML

- Definimos `drawable/rectangulo.xml`

```
<shape xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
  <solid android:color="#f00"/>
  <stroke android:width="2dp" android:color="#00f"
    android:dashWidth="10dp" android:dashGap="5dp"/>
</shape>
```

- Lo utilizamos en un componente `Button`

```
<Button android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/rectangulo" />
```

Drawables en código Java

- Definir el objeto `Drawable`

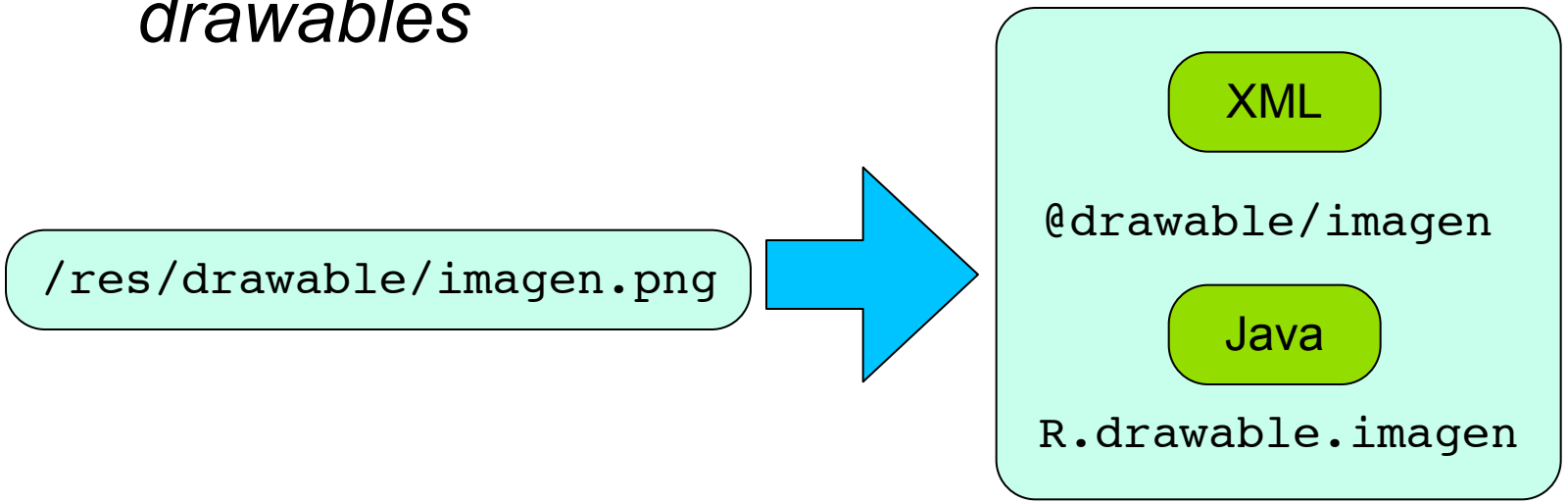
```
RectShape r = new RectShape();
ShapeDrawable sd = new ShapeDrawable(r);
sd.getPaint().setColor(Color.RED);
sd.setIntrinsicWidth(100);
sd.setIntrinsicHeight(50);
```

- Mostrar en un componente

```
ImageView visor = (ImageView)findViewById(R.id.visor);
visor.setImageDrawable(sd);
```

Imágenes

- Basta con introducirlas en el directorio de *drawables*



- Formatos GIF, JPEG y PNG (preferible)
- Android SDK las optimiza automáticamente para reducir el espacio ocupado

Ejemplos de uso comunes

- En un componente `ImageView`

```
<ImageView android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:src="@drawable/imagen" />
```

- Icono de un botón


```
<Button android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/icono" />
```

http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

Imágenes *nine-patch*

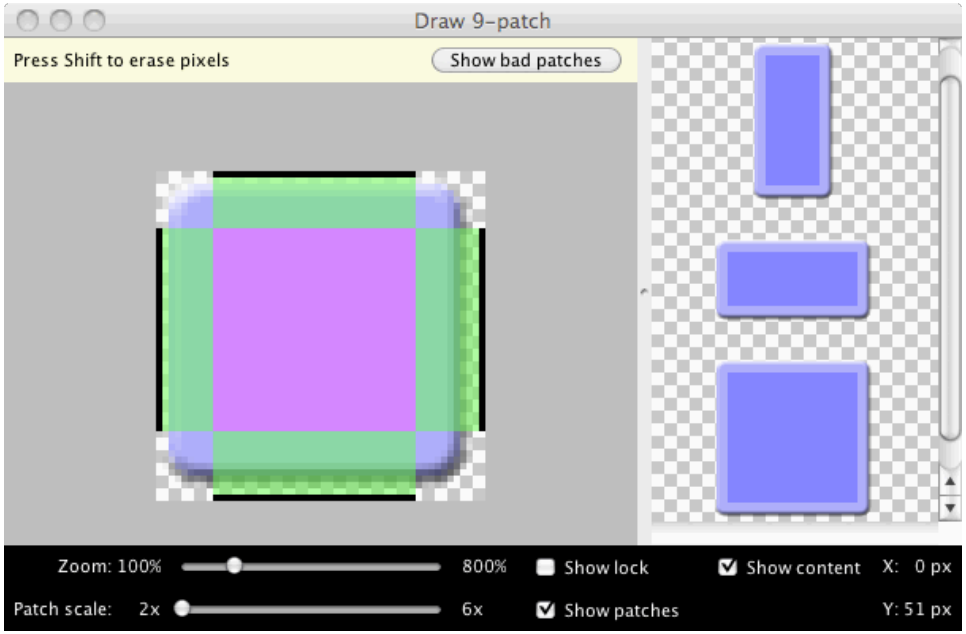
- Muchos componentes no tienen tamaño fijo
- Si su aspecto se define como una imagen aparecerá estirada

- Solución: imágenes *nine-patch*

- Sólo se estiran las regiones centrales

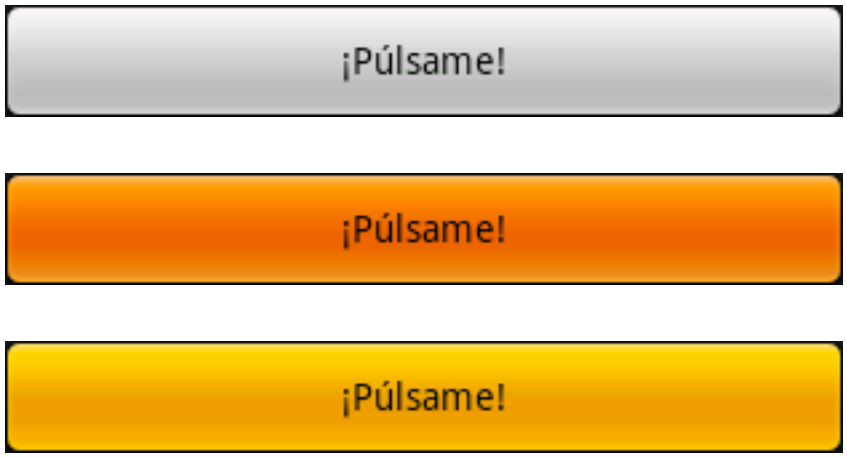
Herramienta `draw9patch`

- En `$ANDROID_SDK_HOME/tools/draw9patch`
- Arrastrar el PNG a la ventana
- Barras superior e izquierda definen área “estirable”
- Barras inferior y derecha definen área de contenido



Personalización de botones

- Los botones no siempre muestran la misma imagen
- Hay varios estados



normal

focused

pressed

- Se puede tratar con *state list drawable*

State list drawable

- Se define un *drawable* para cada estado

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
          "http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/boton_pressed" />
    <item android:state_focused="true"
          android:drawable="@drawable/boton_selected" />
    <item android:drawable="@drawable/boton_normal" />
</selector>
```

http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList

Animaciones por fotogramas en XML

- Se trata de un *drawable* animado
- Lista de fotogramas (*drawables*) y duración

```
<animation-list
        xmlns:android=
        "http://schemas.android.com/apk/res/android"
        android:oneshot="false">
  <item android:drawable="@drawable/frame0
        android:duration="50" />
  <item android:drawable="@drawable/frame1"
        android:duration="50" />
  <item android:drawable="@drawable/frame2"
        android:duration="50" />
</animation-list>
```

Se repite indefinidamente

Mostrar animación por fotogramas

- La reproducción es automática en algunos componentes, por ejemplo `ProgressBar`

- En otros debemos ponerla en marcha
  - Obtener *drawable* del componente

```
ImageView iv = (ImageView) findViewById(R.id.visor);
AnimationDrawable ad = (AnimationDrawable) iv.getBackground();
```

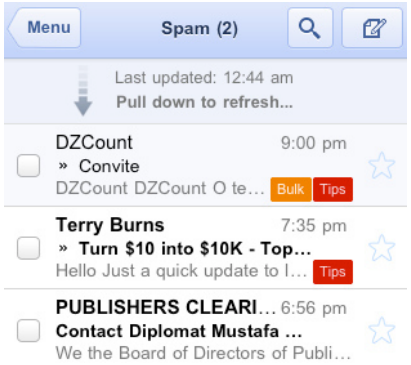
  - Ejecutar `ad.start();`
  - Detener `ad.stop();`

No se puede hacer en `onCreate`

Crear componentes propios

- Comprobar si existen componentes similares
- Podemos heredar de ellos y aprovechar parte de su funcionalidad

Segmented Control → `RadioButton`



Pull to refresh list → `ListView`

- Si ninguno se ajusta, heredamos de `View`

Gráficos en componentes propios

- Crear nuevo subtipo de `View`
- Sobrescribir el método `onDraw(Canvas c)`
  - Recibe parámetro `Canvas` (lienzo)

```
public class MiVista extends View {
    public MiVista(Context context) {
        super(context);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        // TODO Definir como dibujar el componente
    }
}
```

Lienzo y pincel

- Pintamos en el lienzo (`Canvas`)
  - Área de dibujo de nuestro componente
  - Tiene un tamaño (área que ocupa de la pantalla)
  - Define área de recorte, transformaciones, etc
- Usamos un pincel (`Paint`) para pintar en él
- Define la forma en la que se dibuja
  - Color del pincel
  - Tipo de trazo
  - Otros efectos

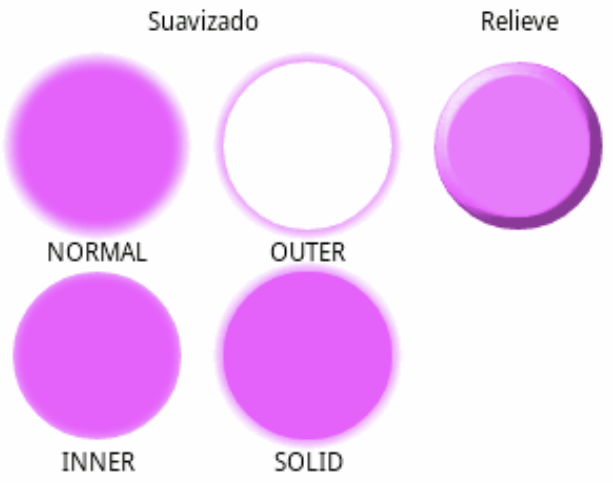
```
Paint p = new Paint();
p.setColor(Color.RED);
```

Atributos del pincel

Estilo del pincel

Color sólido o gradiente

Máscaras



Tipo de trazo

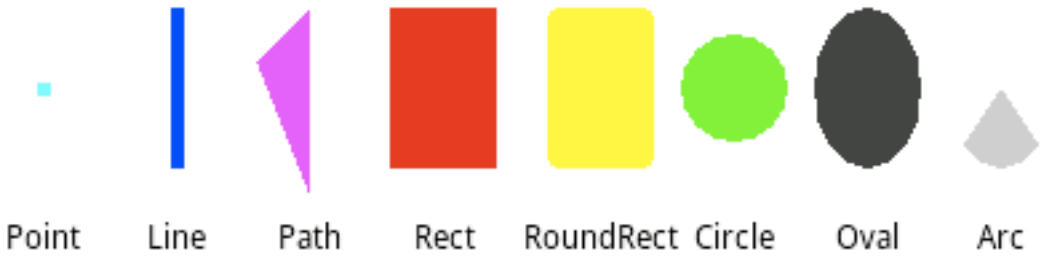
Dithering

Primitivas geométricas

- Se dibujan con métodos de `Canvas`

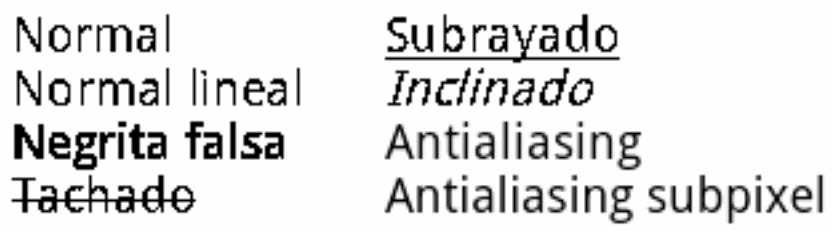
```
Paint paint = new Paint();
paint.setStyle(Style.FILL);
paint.setStrokeWidth(5);
paint.setColor(Color.BLUE);

canvas.drawPoint(100, 100, paint);
canvas.drawLine(10, 300, 200, 350, paint);
canvas.drawRect(new RectF(180, 20, 220, 80), paint);
```



Texto

- Establecer atributos del texto en el pincel



- Dibujar texto en el lienzo (`drawText`)
- Métricas
  - Mide texto en pixeles
  - Separación recomendada entre líneas
  - Anchura de una cadena

Imágenes

- Clase `Bitmap`
  - Se dibujan en el lienzo con `drawBitmap`
  - Liberar memoria con `recycle`
- Inmutables
  - No se puede modificar su contenido
  - `BitmapFactory` para leer diferentes fuentes
    - Ficheros, *arrays* de pixels, recursos, flujos de entrada
- Mutables
  - Podemos modificar su contenido en el código
  - Se crean vacías, proporcionando ancho y alto

Medición del componente

- Se debe ajustar al espacio cedido por el *layout*
- Sobrescribir `onMeasure` para ajustar tamaño
- Recibimos para ancho y alto
  - Tamaño en píxeles
  - Modo

`EXACTLY`	Nos dan el tamaño exacto
`AT_MOST`	Nos dan el tamaño máximo
`UNSPECIFIED`	Sin restricciones

- Debemos establecer el tamaño concreto

```
setMeasuredDimension(width, height)
```

Declaración en el XML

- Utilizamos nombre completo como etiqueta

```
<es.ua.jtech.grafica.GraficaView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
 />
```

- Alternativa

```
<view
    class="es.ua.jtech.grafica.GraficaView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
 />
```

Atributos propios

- Puede que necesitemos parametrizar el componente
- Declaramos nuestros propios atributos
- En `/res/values/attrs.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Grafica">
        <attr name="percentage" format="integer"/>
    </declare-styleable>
</resources>
```

Uso de atributos propios

- Debemos declarar el espacio de nombres

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
     "http://schemas.android.com/apk/res/android"
    xmlns:app=
     "http://schemas.android.com/apk/res/es.ua.jtech.grafica"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<es.ua.jtech.grafica.GraficaView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:percentage="60"
 />
</LinearLayout>
```

Paquete declarado en el *manifest*

Lectura de los atributos

- Definir los constructores a partir de atributos

```
public GraficaView(Context context, AttributeSet attrs, int defStyle) {
  super(context, attrs, defStyle);
  this.init(attrs);
}

public GraficaView(Context context, AttributeSet attrs) {
  super(context, attrs);
  this.init(attrs);
}
```

- Leer los atributos

```
private void init(AttributeSet attrs) {
  TypedArray ta = this.getContext().obtainStyledAttributes(attrs,
                                                R.styleable.Grafica);
  this.percentage = ta.getInt(R.styleable.Grafica_percentage, 0);
}
```

Actualización del contenido

- Cambiar propiedades de objetos de la vista
- Llamar a `invalidate` para que se repinte
- Si usamos un hilo diferente al hilo UI llamar a `postInvalidate`

- Esto no es apropiado para elevadas tasas de refresco
- Si necesitamos animaciones fluidas, utilizaremos `SurfaceView`

Gráficos 3D

- `View` es útil para mostrar gráficos sencillos
- Es poco eficiente para
  - Gráficos 3D
  - Tasas elevadas de refresco
- Para aplicaciones con alta carga gráfica
  - Utilizaremos `SurfaceView`
  - Se dibuja en hilo independiente
  - No bloquea hilo principal de eventos
  - OpenGL para gráficos 3D
  - A partir de 1.5 tenemos `GLSurfaceView`

SurfaceView

```
public class VistaSurface extends SurfaceView
                implements SurfaceHolder.Callback {
    HiloDibujo hilo = null;
    public VistaSurface(Context context) {
        super(context);
        SurfaceHolder holder = this.getHolder();
        holder.addCallback(this);
    }
    public void surfaceChanged(SurfaceHolder holder, int format,
                               int width, int height) {
        // La superficie ha cambiado (formato o dimensiones)
    }
    public void surfaceCreated(SurfaceHolder holder) {
        hilo = new HiloDibujo(holder, this);
        hilo.start();
    }
    public void surfaceDestroyed(SurfaceHolder holder) {
        // Detener hilo
    }
}
```

Heredamos de `SurfaceView` e implementamos `SurfaceHolder.Callback`

Obtenermos el *holder* de la superficie y registramos el *callback*

Al crearse la superficie ejecutamos el hilo de dibujo

Al destruirse lo paramos

Hilo de dibujo

```
public void run() {
    while (continuar) {
        Canvas c = null;
        try {
            c = holder.lockCanvas(null);
            synchronized (holder) {
                // Dibujar aqui los graficos
                c.drawColor(Color.BLUE);
            }
        } finally {
            if (c != null) {
                holder.unlockCanvasAndPost(c);
            }
        }
    }
}
```

Obtenemos el lienzo a partir del *holder*, y lo bloqueamos

Debemos dibujar de forma sincronizada con el *holder*

Desbloqueamos el lienzo y mostramos en pantalla lo dibujado

GLSurfaceView

- Se encarga de:
  - Inicialización y destrucción del contexto OpenGL
  - Gestión del hilo de *render*
- No hace falta sobrescribir la clase
- Debemos definir un objeto `Renderer`

```
public class MiRenderer implements GLSurfaceView.Renderer {

    public void onSurfaceCreated(GL10 gl,
                                 EGLConfig config) { ... }
    public void onSurfaceChanged(GL10 gl, int w,
                                 int h) { ... }
    public void onDrawFrame(GL10 gl) { ... }
}
```

Creación de la vista

```
public class MiActividad extends Activity {
    GLSurfaceView vista;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        vista = new GLSurfaceView(this);
        vista.setRenderer(new MiRenderer());
        setContentView(vista);
    }
    @Override
    protected void onPause() {
        super.onPause();
        vista.onPause();
    }
    @Override
    protected void onResume() {
        super.onResume();
        vista.onResume();
    }
}
```

Proporcionamos nuestro *renderer*

Comunicamos a la vista de OpenGL los eventos de pausa y reanudación

Desarrollo de videojuegos

- Existen librerías que facilitan el trabajo
  - Gestionan los gráficos OpenGL
  - Proporcionan el hilo del juego
  - Carga de modelos 3D (OBJ, MD2, etc)

- Destacamos
  - AndEngine
    - http://www.andengine.org/
  - libgdx
    - http://libgdx.badlogicgames.com/

¿Preguntas...?