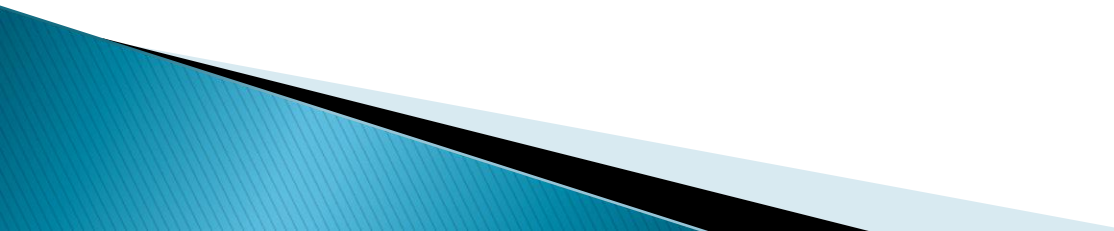


# Android: Uso de gráficos

# Recursos gráficos

Para dibujar en Android se dispone de los siguientes recursos:

- ▶ Creando un clase que extienda la clase View.
  - ▶ Creando una clase que extienda la clase SurfaceView.
  - ▶ Usando OpenGL ES.
- 

# View – vista

Crear una clase que se extienda de la clase View.

Deberemos implementar su constructor y, al menos, sobrescribir el método **public void onDraw(Canvas c)**.

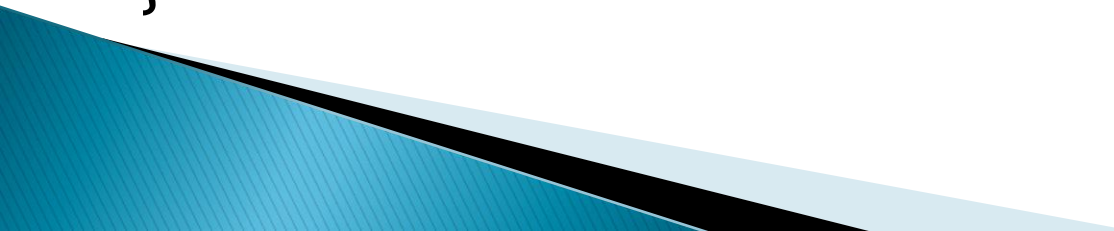
# Canvas – lienzo

- ▶ El canvas (lienzo) es la superficie sobre la que se dibuja.
- ▶ La forma más sencilla y directa de dibujar consiste en utilizar los métodos de la clase Canvas.

# Paint – pincel

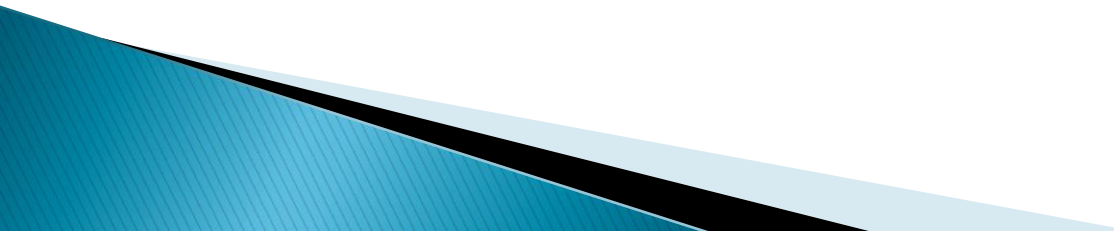
- ▶ La clase Paint permite definir la forma y los colores con los que se dibuja sobre el lienzo.

```
public class Vista extends View {  
    public Vista(Context contexto) {  
        super(contexto);  
    }  
    @Override  
    public void onDraw(Canvas lienzo) {  
        Paint pincel = new Paint();  
        pincel.setColor(Color.GRAY);  
        pincel.setAntiAlias(true);  
        lienzo.drawRect(x1, y1, x2, y2, pincel);  
    }  
}
```



# Instanciación

```
public class Graficos extends Activity {  
    @Override  
    protected void onCreate(Bundle estado) {  
        super.onCreate(estado);  
        setContentView(new Vista(this));  
    }  
}
```



# Dimensiones del View

@Override

```
public void onSizeChanged(int w, int h, int oldw, int oldh) {  
    super.onSizeChanged(w, h, oldw, oldh);  
    ancho = w;  
    alto = h;  
}
```

Podemos obtener las dimensiones del control sobrescribiendo el método onSizeChanged








# Dibujar de forma interactiva

Este evento permite registrar todas las interacciones que se producen sobre la vista.

Importante:

**return true;**

**@Override**

```
public boolean onTouchEvent(MotionEvent event) {  
    float x = event.getX();  
    float y = event.getY();  
    switch (event.getAction()) {  
        case MotionEvent.ACTION_DOWN:  
            ...  
            break;   
        case MotionEvent.ACTION_MOVE:  
            ...  
            break;   
        case MotionEvent.ACTION_UP:  
            ...  
            break;   
    }  
    return true;  
}
```

Empiezo a Pulsar

Nuevo

Dejar de Pulsar

# Acciones

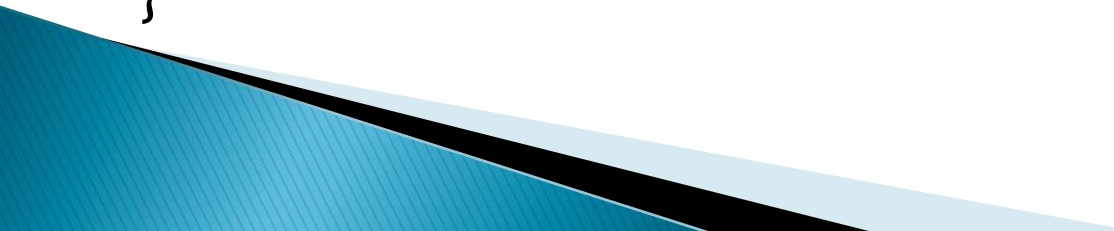
- ▶ Debemos distinguir cuando se empieza a dibujar (**ACTION\_DOWN**), cuando se está dibujando (**ACTION\_MOVE**) y cuando se deja de dibujar (**ACTION\_UP**).
- ▶ Debemos evitar que el evento sea procesado por otro gestor de eventos: **return true;**.

# Trazar una recta de forma interactiva

- ▶ Al empezar a dibujar guardamos la coordenada inicial en  $x_0$  e  $y_0$ .
- ▶ Mientras dibujamos, guardamos la coordenada en  $x_i$  e  $y_i$  e invalidamos la vista anterior.
- ▶ Al dejar de dibujar, guardamos la coordenada en  $x_i$  e  $y_i$  e invalidamos la vista anterior.
- ▶ En `onDraw` dibujamos una recta desde  $x_0$ ,  $y_0$  a  $x_i$ ,  $y_i$ .

@Override

```
public boolean onTouchEvent(MotionEvent event) {  
    float x = event.getX();  
    float y = event.getY();  
    switch (event.getAction()) {  
        case MotionEvent.ACTION_DOWN:  
            x0=x;  
            y0=y;  
            break;  
        case MotionEvent.ACTION_MOVE:  
        case MotionEvent.ACTION_UP:  
            xi=x;  
            yi=y;  
            invalidate();  
    }  
    return true;  
}
```



# onDraw()

De este modo veremos que conforme nos vamos moviendo sobre la superficie del control, se va trazando la recta que dibujamos. Al dejar de dibujar permanecerá la recta sobre el control, hasta que empecemos a dibujar otra.

@Override

```
public void onDraw(Canvas lienzo) {  
    lienzo.drawLine(x0, y0, xi, yi, pincel);  
}
```

# “Guardar” los elementos dibujados

Para no perder los elementos pintados con anterioridad, podemos seguir una de las siguientes estrategias.

1. Guardamos en una estructura de datos dinámica, por ejemplo, un ArrayList, la sucesión de rectas que vamos acumulando, de este modo en onDraw, primero recorreremos el ArrayList y dibujamos todas las rectas anteriores y después la recta actual.
2. Guardamos los elementos dibujados en nuestro propio lienzo con ayuda de un mapa de bits, de modo que en onDraw, primero dibujamos nuestro lienzo y después la recta actual.
3. Si queremos poder “deshacer”, tendremos que utilizar una mezcla de ambas estrategias.

# Uso del Bitmap

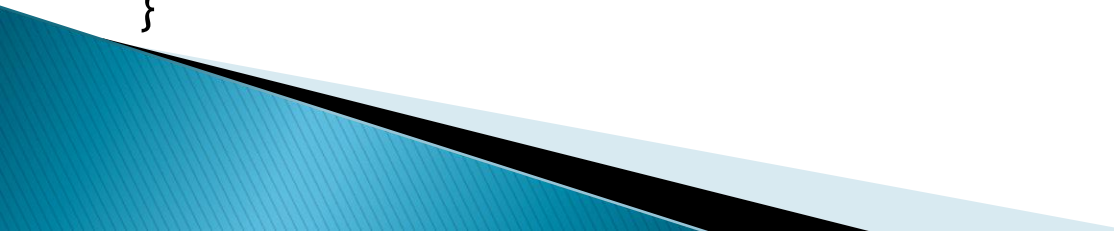
```
private Bitmap mapaDeBits;  
private Canvas lienzoFondo;
```

```
@Override
```

```
public void onSizeChanged(int w, int h, int oldw, int oldh) {  
    super.onSizeChanged(w, h, oldw, oldh);  
    mapaDeBits = Bitmap.createBitmap(w, h,  
        Bitmap.Config.ARGB_8888);  
    lienzoFondo = new Canvas(mapaDeBits);  
}
```

```
@Override
```

```
public void onDraw(Canvas lienzo) {  
    lienzo.drawBitmap(mapaDeBits, 0, 0, null);  
    lienzo.drawLine(x0, y0, xi, yi, pincel);  
}
```



# Agregar la recta al lienzo

Cuando se produzca el evento  
`MotionEvent.ACTION_UP`:

```
lienzoFondo.drawLine(x0, y0, xi, yi, pincel);
```



# Trazar un círculo de forma interactiva

El método `drawCircle(x0, y0, radio, pincel)` permite dibujar un círculo. `x0, y0` son la coordenada inicial el radio lo obtenemos a partir de la distancia que hay con respecto al punto actual `xi, yi`:

```
(float)Math.sqrt(Math.pow(x0-xi,2)+Math.pow(y0-yi,2))
```

@Override

```
public void onDraw(Canvas lienzo) {  
    lienzo.drawBitmap(mapaDeBits, 0, 0, null);  
    lienzo.drawCircle(x0, y0, radio, pincel);  
}
```

# Trazar una recta poligonal

- ▶ Down: guardar la coordenada inicial  $x_0, y_0$ .
- ▶ Move: guardar la coordenada  $x_i, y_i$ , trazar recta de  $x_0, y_0$  a  $x_i, y_i$ . Asignar  $x_0 = x_i, y_0 = y_i$ .
- ▶ Up: guardar la coordenada  $x_i, y_i$ , trazar recta de  $x_0, y_0$  a  $x_i, y_i$ .
- ▶ Todas las rectas se van trazando directamente sobre el lienzo del fondo.

# Path I

La clase Path permite dibujar rectas y curvas.

```
private Path rectaPoligonal = new Path();  
...  
pincel.setStyle(Paint.Style.STROKE);
```

Al empezar:

```
x0 = xi = event.getX();  
y0 = yi = event.getY();  
rectaPoligonal.reset();  
rectaPoligonal.moveTo(x0, y0);
```

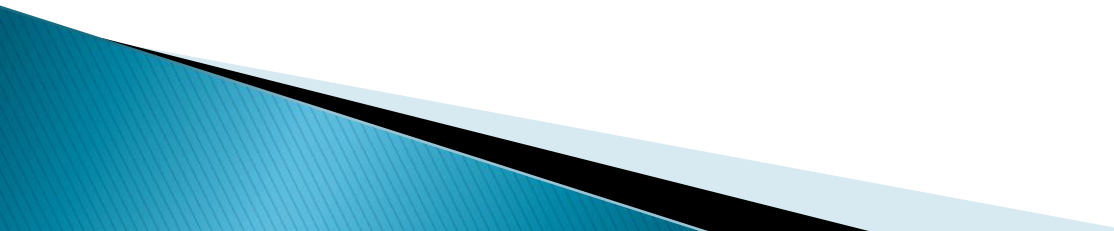
# Path II

Mientras se dibuja:

```
rectaPoligonal.quadTo(xi, yi,  
                      (x + xi) / 2, (y + yi)/2);  
xi = x;  
yi = y;
```

Al finalizar:

```
lienzoFondo.drawPath(rectaPoligonal, pincel);  
rectaPoligonal.reset();
```



# Path III

@Override

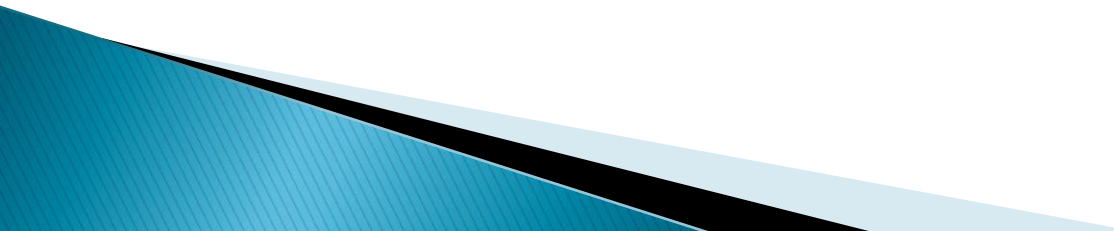
```
public void onDraw(Canvas lienzo) {  
    lienzo.drawBitmap(mapaDeBits, 0, 0, null);  
    lienzo.drawPath(rectaPoligonal, pincel);  
}
```

# Guardar un Bitmap

```
File carpeta=new File(Environment.getExternalStorageDirectory().  
    getPath());  
File archivo = new File (carpeta, nombre);  
FileOutputStream fos = new FileOutputStream (archivo);  
Bitmap.createBitmap(ancho, alto,  
    Bitmap.Config.ARGB_8888).compress(  
    CompressFormat.PNG, 90, fos);
```

# Cargar un Bitmap

```
File carpeta=new  
File(Environment.getExternalStorageDirectory().getPath());  
File archivo = new File (carpeta, nombre);  
imagenFondo = Bitmap.createBitmap(ancho, alto,  
    Bitmap.Config.ARGB_8888);  
if (archivo.exists()){  
    BitmapFactory.Options options = new BitmapFactory.Options();  
    options.inMutable=true;  
    imagenFondo=BitmapFactory.decodeFile(  
        archivo.getAbsolutePath(),options);  
}  
lienzoFondo = new Canvas(imagenFondo);
```



# Seleccionar un color

- ▶ Constructor:  

```
public ColorPickerDialog(  
    Context context,  
    OnColorChangeListener listener,  
    int initialColor)
```
- ▶ `ColorPickerDialog.OnColorChangeListener`:  

```
@Override  
public void colorChanged(int color) {  
    ...  
}
```
- ▶ Mostrar:  

```
new ColorPickerDialog(...).show();
```