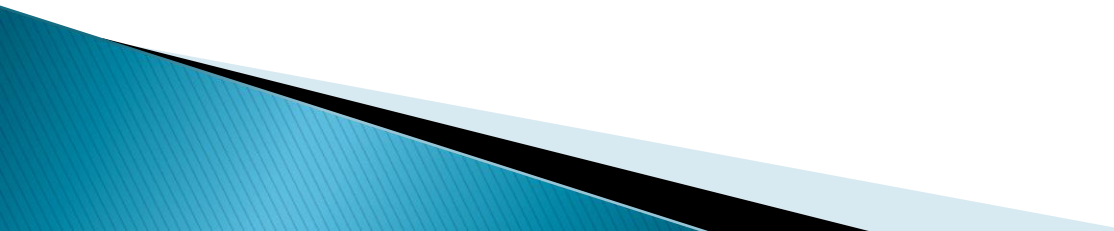


# Desarrollo de un juego

`android.view.SurfaceView`

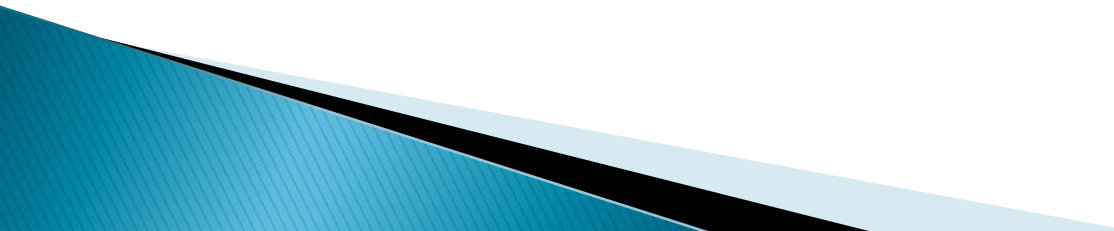
# Actividad

```
VistaJuego vj;  
@Override  
protected void onCreate(Bundle  
    savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    vj = new VistaJuego(this);  
    setContentView(vj);  
}
```



# Constructor

```
public class VistaJuego extends SurfaceView {  
    private SurfaceHolder contenedorSuperficie;  
    private Bitmap bmp;  
    private int alto, ancho;  
    public VistaJuego(Context context) {  
        super(context);  
        contenedorSuperficie = getHolder();  
        GestionarSuperficie gestor = new GestionarSuperficie();  
        contenedorSuperficie.addCallback(gestor);  
        bmp = BitmapFactory.decodeResource(  
            getResources(), R.drawable.ic_launcher);  
    }  
}
```

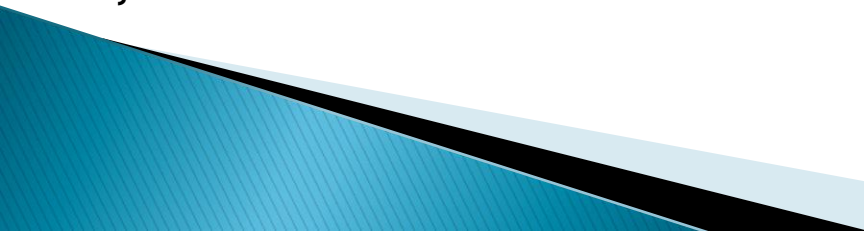


# Clases usadas

- ▶ SurfaceView, similar a View, proporciona un acceso más directo y rápido al dispositivo.
- ▶ SurfaceHolder, contenedor del SurfaceView, se usa para gestionar las eventos que se producen sobre el SurfaceView.
- ▶ GestionarSuperficie implements SurfaceHolder.Callback, es el objeto gestor de los eventos que se producen sobre el SurfaceView

# SurfaceHolder.Callback

```
private class GestionarSuperficie implements SurfaceHolder.Callback {  
    @Override  
    public void surfaceChanged (SurfaceHolder holder, int format, int width,  
                                int height) {  
        alto=height;  
        ancho=width;  
    }  
    @Override  
    public void surfaceCreated (SurfaceHolder holder) {  
        Canvas lienzo = contenedorSuperficie.lockCanvas(null);  
        onDraw(lienzo);  
        contenedorSuperficie.unlockCanvasAndPost(lienzo);  
    }  
    @Override  
    public void surfaceDestroyed (SurfaceHolder holder) {  
    }  
}
```



# surfaceCreated()

Para dibujar en la SurfaceView se siguen estos pasos:

- ▶ se obtiene el canvas al tiempo que se bloquea
- ▶ se dibuja (onDraw)
- ▶ se desbloquea el canvas

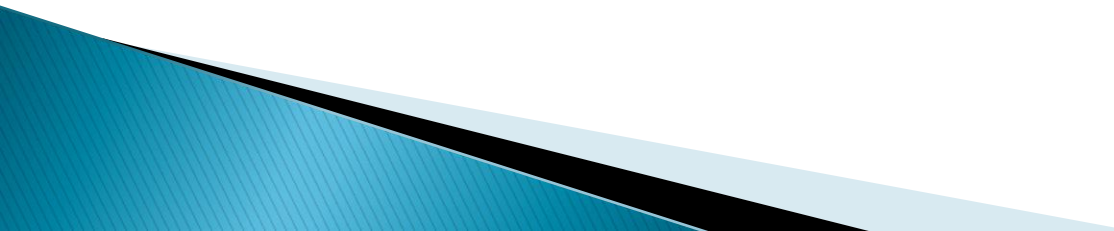
# onDraw()

@Override

```
protected void onDraw(Canvas canvas) {  
    canvas.drawBitmap(bmp, 10, 10, null);  
}
```

# Hebra del juego

```
public class HebraJuego extends Thread {  
    private VistaJuego vista;  
    private boolean funcionando = false;  
    private static final long FPS = 10;  
    public HebraJuego(VistaJuego vj) {  
        this.vista = vj;  
    }  
    public void setFuncionando(boolean f) {  
        funcionando = f;  
    }  
}
```



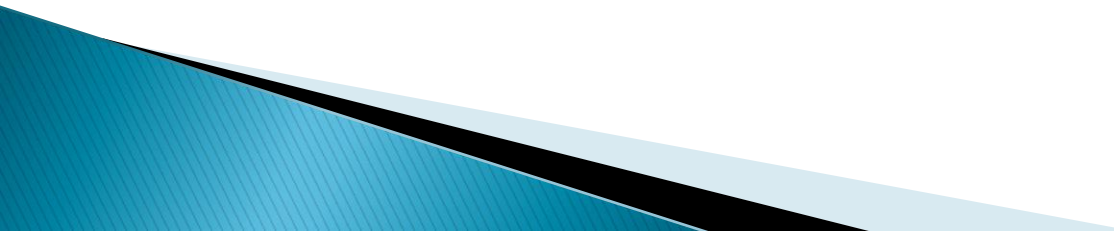


# run()

```
@Override
public void run() {
    long inicio;
    long ticksPS = 1000 / FPS;
    long tiempoEspera;
    while (funcionando) {
        Canvas canvas = null;
        inicio = System.currentTimeMillis();
        try {
            canvas = vista.getHolder().
                           lockCanvas();
            synchronized (vista.getHolder()) {
                vista.onDraw(canvas);
            }
        }
    }
}
```

```
finally {
    if (canvas != null) {
        vista.getHolder().
            unlockCanvasAndPost(canvas);
    }
}
tiempoespera = ticksPS -
               (System.currentTimeMillis() - inicio);
try {
    if (tiempoespera > 0)
        sleep(tiempoespera);
    else
        sleep(10);
} catch (InterruptedException e) {
}
}
```

# Funcionamiento de run()

- ▶ Se obtiene acceso al canvas.
  - ▶ Se dibuja el canvas, de forma sincronizada, para evitar que otra hebra dibuje el canvas al mismo tiempo.
  - ▶ Se desbloquea el canvas.
  - ▶ Se calcula el tiempo que falta para volver a dibujar el canvas. La hebra se duerme el tiempo necesario.
- 

# Cambios en la SurfaceView I

- ▶ Variables de instancia:

```
private HebraJuego hebraJuego;
```

```
private int ejeY = 0;
```

```
private int direccionY;
```

- ▶ Constructor:

```
public VistaJuego(Context context) {
```

```
    ...
```

```
    hebraJuego = new HebraJuego(this);
```

```
}
```



# Cambios en la SurfaceView II

@Override

```
public void surfaceCreated(SurfaceHolder holder) {  
    hebraJuego.setFuncionando(true);  
    hebraJuego.start();  
}
```

@Override

```
public void surfaceDestroyed(SurfaceHolder holder) {  
    boolean reintentar = true;  
    hebraJuego.setFuncionando(false);  
    while (reintentar) {  
        try {  
            hebraJuego.join();  
            reintentar = false;  
        } catch (InterruptedException e) {  
        }  
    }  
}
```

# Cambios en onDraw()

@Override

```
protected void onDraw(Canvas canvas) {  
    canvas.drawColor(color);  
    if (ejeY >= getHeight() - bmp.getHeight()) {  
        direccionY = -10;  
    } else if (ejeY <= 0) {  
        direccionY = 10;  
    }  
    ejeY = ejeY + direccionY;  
    canvas.drawBitmap(bmp, 10, ejeY, null);  
}
```

# Eje X

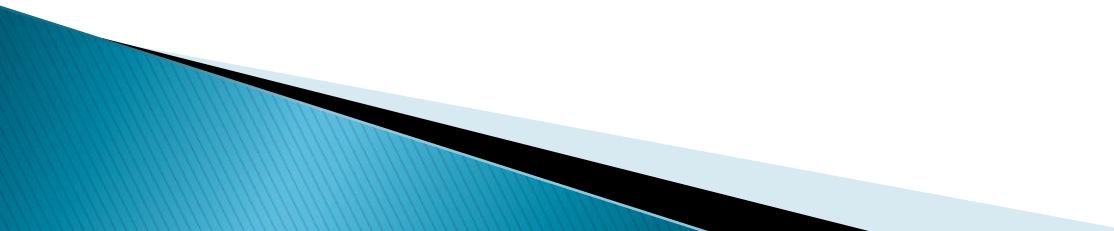
```
private int ejeX = 0;
private int direccionX;
if (ejeX >= getWidth() - bmp.getWidth()) {
    direccionX = -10;
} else if (ejeX <= 0) {
    direccionX = 10;
}
ejeX = ejeX + direccionX;
canvas.drawBitmap(bmp, ejeX, ejeY, null);
```

# Clase Figura

- Implementar una clase en la que se controle el funcionamiento de los desplazamientos de la figura.

```
private Figura f=null;
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.DKGRAY);
    if(f==null)
        f=new Figura(this, bmp);
    f.dibujar(canvas);
}
```

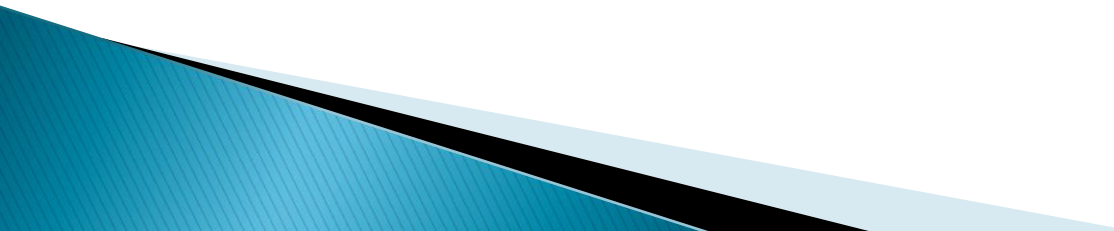
# Parámetros de interés

- ▶ Número de pixel de un control: `getWidth()` y `getHeight()`;
  - ▶ Densidad de la pantalla: `getContext().getResources().getDisplayMetrics().density`.
  - ▶ Número de pixel a lo alto del dispositivo: `getResources().getDisplayMetrics().heightPixels`.
  - ▶ Número de pixel a lo ancho del dispositivo: `getResources().getDisplayMetrics().widthPixels`.
- 



# Implementación clase Figura

```
public class Figura {  
    private VistaJuego vista;  
    private Bitmap bmp;  
    private int ancho, alto;  
    private int ejeX, ejeY;  
    private int direccionX, direccionY;  
}
```



# Constructor

```
public Figura(VistaJuego vista, Bitmap bmp) {  
    this.vista = vista;  
    this.bmp = bmp;  
    this.anch = bmp.getWidth();  
    this.alto = bmp.getHeight();  
    Random rnd = new Random();  
    ejeX = rnd.nextInt(vista.getWidth() - this.anch);  
    ejeY = rnd.nextInt(vista.getHeight() - this.alto);  
    direccionX = rnd.nextInt(12) - 5;  
    if(direccionX == 0) direccionX = 1;  
    direccionY = rnd.nextInt(12) - 5;  
    if(direccionY == 0) direccionY = 1;  
}
```

# Método dibujar

```
public void dibujar(Canvas canvas) {  
    movimiento();  
    canvas.drawBitmap bmp, ejeX, ejeY, null);  
}
```

# Método movimiento

```
private void movimiento(){  
    if (ejeX > vista.getWidth() - ancho - direccionX ||  
        ejeX + direccionX < 0) {  
        direccionX = -direccionX;  
    }  
    ejeX = ejeX + direccionX;  
    if (ejeY > vista.getHeight() - alto - direccionY ||  
        ejeY + direccionY < 0) {  
        direccionY = -direccionY;  
    }  
    ejeY = ejeY + direccionY;  
}
```

# Interacción: touch

```
private long ultimoClick = 0;
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (System.currentTimeMillis() - ultimoClick > 300) {
        ultimoClick = System.currentTimeMillis();
        float x,y;
        x=event.getX();
        y=event.getY();
        synchronized (getHolder()) {
            if(f.tocado(x, y)){
                f=new Figura(this, bmp);
            }
        }
    }
    return true;
}
```

# Método tocado

- ▶ Usamos el ultimoClic para evitar que el dedo se quede permanentemente pulsado.
- ▶ Se comprueba la condición de forma sincronizada, para evitar que la figura sea desplazada durante la comprobación.

```
public boolean tocado(float x, float y){  
    return x > ejeX && x < ejeX + ancho &&  
           y > ejeY && y < ejeY + alto;  
}
```

# Sprites

- ▶ Un sprite es una imagen que contiene varias imágenes. Se suelen utilizar para simular pequeñas animaciones.



# Clase Sprite

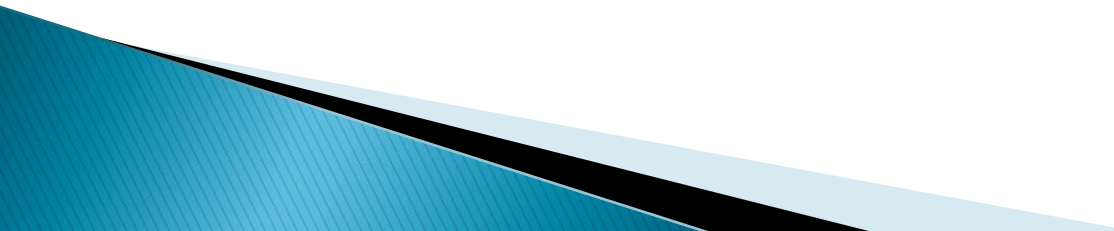
```
public class Sprite {  
    private Bitmap bmp; // imagen sprite  
    private int frameActual=0;  
    private static final int COLUMNAS = 4;  
    private static final int FILAS = 2;  
    public Sprite(VistaJuego vista, Bitmap bmp) {  
        this.alto=bmp.getHeight()/FILAS;  
        this.ancha=bmp.getWidth()/COLUMNAS;  
    }  
}
```



# Dibujo y movimiento

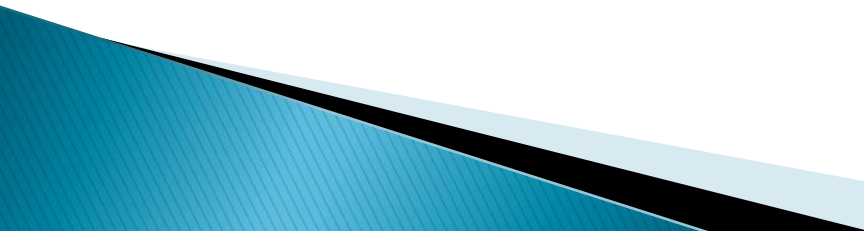
```
private void movimiento(){
    frameActual = ++frameActual % COLUMNAS;
}
public void dibujar(Canvas canvas) {
    movimiento();
    int origenx = frameActual * ancho;
    int origeny = 0;
    if(direccionX<0)
        origeny=alto;
    else
        origeny=0;
    Rect origen = new Rect(origenx, origeny, origenx + ancho, origeny +
        alto);
    Rect destino = new Rect(ejeX, ejeY, ejeX + ancho, ejeY + alto);
    canvas.drawBitmap(bmp, origen, destino, null);
}
```

# Otros métodos

- ▶ **public void setPosicion(float x, float y)**
  - ▶ **public void setMovimiento(float x, float y)**
  - ▶ **etc.**
- 

# Velocidad del movimiento I

```
private VelocityTracker controlVelocidad = null;
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            if(controlVelocidad == null) {
                controlVelocidad = VelocityTracker.obtain();
            }
            else {
                controlVelocidad.clear();
            }
            controlVelocidad.addMovement(event);
            ...
    }
}
```



# Velocidad del movimiento II

```
case MotionEvent.ACTION_MOVE:
    controlVelocidad.addMovement(event);
    controlVelocidad.computeCurrentVelocity(1000);
    velX = VelocityTrackerCompat.getXVelocity(
        controlVelocidad,
        event.getPointerId(event.getActionIndex()));
    velY = VelocityTrackerCompat.getYVelocity(
        controlVelocidad,
        event.getPointerId(event.getActionIndex()));
    ...
```

