24. 2D Graphics and Animation

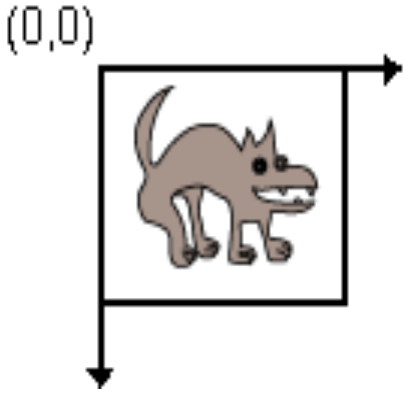
Drawing 2D graphics

- To draw our own custom 2D graphics on screen,
  we'll make a **custom View subclass** with the drawing code.

- If the app is animated (such as a game), we'll also use a **thread**
  to periodically update the graphics and redraw them.

Custom View template

```
public class ClassName extends View {
    // required constructor
    public ClassName(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    // this method draws on the view
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        drawing code;
    }
}
    // recall: y-axis increases downward!
```



Using your custom view

- You can insert your custom view into an activity's layout XML:

```
<!-- res/layout/activity_main.xml -->
<RelativeLayout ...
    tools:context=".MainActivity">

    <packageName.ClassName
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ...
    />

</RelativeLayout>
```

Canvas object methods

Method	Description
`drawARGB(alpha, r, g, b);`	fill window with color (rgb=0-255)
`drawArc(...);`	draw a partial ellipse
`drawBitmap(bmp, x, y, null);`	draw an image
`drawCircle(centerX, centerY, r, paint);`	draw a circle
`drawLine(x1, y1, x2, y2, paint);`	draw a line segment
`drawOval(x1, y1, x2, y2, paint);` * `drawOval(new RectF(x1, y1, x2, y2), paint);`	draw oval/circle
`drawPoint(x, y, paint);`	color a single pixel
`drawRect(x1, y1, x2, y2, paint);` * `drawRect(new RectF(x1, y1, x2, y2), paint);`	draw rectangle
`drawRoundRect(x1, y1, x2, y2, rx, ry, paint);`* `drawRoundRect(new RectF(x1, y1, x2, y2), rx, ry, paint);`	rounded rectangle
`drawText("str", x, y, paint);`	draw a text string
`getWidth(), getHeight()`	dimensions of view

* = requires Android 5.0+

Paint

- Many methods accept a **Paint**, a color to use for drawing.
  - Create a Paint by specifying an alpha (opacity) value, and red/green/blue (RGB) integer values, from 0 (none) to 255 (full).

```
Paint name = new Paint();
name.setARGB(alpha, red, green, blue);

// example
Paint purple = new Paint();
purple.setARGB(255,255,  0, 255);
purple.setStyle(Style.FILL_AND_STROKE);  // FILL, STROKE
```

Paint methods

Method	Description
`getTextBounds("text", start, end, Rect)`	fill Rect with bounding rectangle that surrounds text
`getTextSize()`	returns text size in px
`getTypeface()`	returns font used
`measureText("text", start, end)`	returns string width
`setAlpha(alpha);`	set color transparency
`setAntiAlias(bool);`	whether to smooth pixels
`setColor(color);`	set paint color as RGB int
`setStrokeWidth(width);`	set line thickness
`setStyle(style);`	set paint styles
`setTextAlign(align);`	sets text alignment
`setTextSize(size);`	sets font size
`setTypeface(typeface);`	sets font

Some common colors

name	red	green	blue
black	0	0	0
blue	0	0	255
brown	139	69	19
cyan	0	255	255
dark gray	64	64	64
gray	128	128	128
light gray	192	192	192
green	0	255	0
orange	255	200	0
pink	255	175	175
purple	255	0	255
red	255	0	0
white	255	255	255
yellow	255	255	0

Typeface

- In Android, a font is called a **Typeface**. Set a font inside a Paint. You can create a Typeface based on a specific font name:

  `Typeface.create("font name", Typeface.STYLE)`

  - styles: NORMAL, BOLD, ITALIC, BOLD_ITALIC
- Or based on a general "font family":

  `Typeface.create(Typeface.FAMILY_NAME, Typeface.STYLE)`

  - family names: DEFAULT, MONOSPACE, SERIF, SANS_SERIF
- Or from a file in your `src/main/assets/` directory:

  `Typeface.createFromAsset(getAssets(), "filename")`

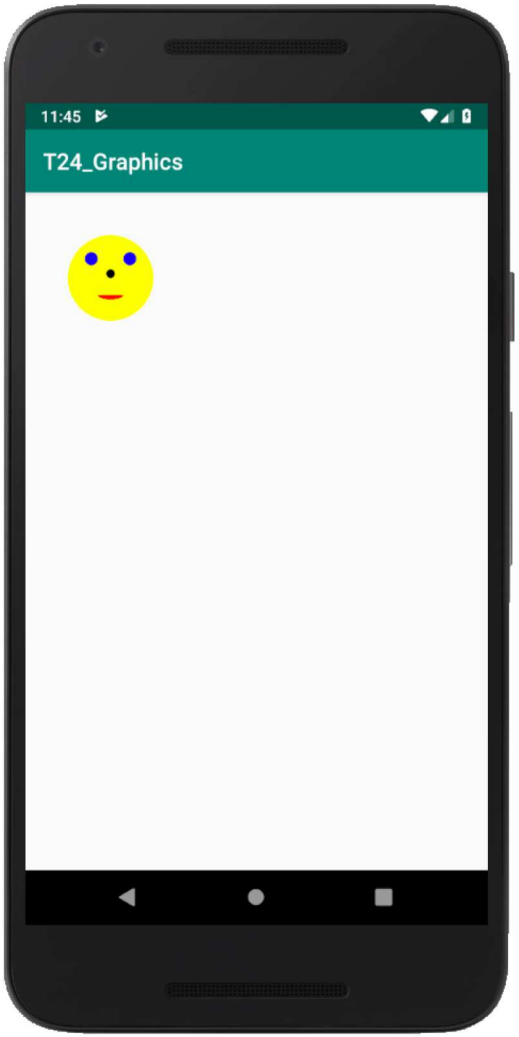
```
// example: use a 40-point monospaced blue font
Paint p = new Paint();
p.setTypeface(
    Typeface.create(Typeface.MONOSPACE, Typeface.BOLD));
p.setTextSize(40);
p.setARGB(255, 0, 0, 255);
```

Smiley face

Write a custom view that draws a "smiley face" figure.

face: (100, 100), size 200, yellow
eyes: (140, 140) and (230, 140), size 30, blue
nose: (190, 180), size 20, black
mouth: (170, 230), size 60x20, red

text: (100, 400), monospaced bold font, size 40



Smiley face

```
public class FaceView extends View { ...
        protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        Paint yellow = new Paint(); // face
        yellow.setARGB(255, 255, 255, 0);
        yellow.setStyle(Paint.Style.FILL_AND_STROKE);
        canvas.drawOval(new RectF(100, 100, 300, 300), yellow);

        Paint blue = new Paint(); // eyes
        blue.setARGB(255, 0, 0, 255);
        blue.setStyle(Paint.Style.FILL_AND_STROKE);
        canvas.drawOval(new RectF(140, 140, 170, 170), blue);
        canvas.drawOval(new RectF(230, 140, 260, 170), blue);

        Paint black = new Paint(); // nose
        black.setARGB(255, 0, 0, 0);
        black.setStyle(Paint.Style.FILL_AND_STROKE);
        canvas.drawOval(new RectF(190, 180, 210, 200), black);

        Paint red = new Paint(); // mouth
        red.setARGB(255, 255, 0, 0); red.setStyle(Paint.Style.FILL_AND_STROKE);
        canvas.drawArc(170, 230, 230, 250, 0, 180, false, red);

        black.setTypeface(Typeface.create(Typeface.MONOSPACE, Typeface.BOLD));
        black.setTextSize(40f); // text
        canvas.drawText("Android is awesome", 100, 400, black); } }
```

Bitmap images

- Draw an image (such as .png or .jpg) using the Bitmap class.

```
Bitmap name = BitmapFactory.decodeResource(
                        getResources(), R.drawable.ID);
```

```
// example:draw  Mario.png on screen at (0, 0)
Bitmap bmp =BitmapFactory.decodeResource(
                        getResources(), R.drawable.Mario);
canvas.drawBitmap(bmp, 0, 0, null);
```

```
// you can also read a Bitmap from an input stream URL
   url =newURL("http://example.com/myImage.jpg");
Bitmap bmp=BitmapFactory.decodeStream(
                        url.openStream());
```

Lib: GCanvas

The Stanford Android library contains a GCanvas class that more easily handles drawing and animation.

http://web.stanford.edu/class/cs193a/lib/

```
public class MyCanvas extends GCanvas { ...
```

The model for GCanvas is different from a regular View:

```
c.drawRect() → GRect, GOval, GLabel
onDraw → init
animation → animate, onAnimateTick
```

GCanvas methods

Method	Description
`add(gobject);` `add(gobject, x, y);`	add graphical object to canvas at top of z-order
`contains(gobject)`	true if this graphical object is in canvas
`getElement(index)`	returns graphical object at given index in list
`getElementAt(x, y)`	top object at given pixel, or null if none
`getElementCount()`	returns number of graphical objects
`init()`	override this to write initialization code
`remove(gobject);`	remove graphical object from canvas
`removeAll();`	removes all graphical objects from canvas

`sendBackward(gobject);`
`sendForward(gobject);`
`sendToBack(gobject);`
`sendToFront(gobject);`
	adjust object's position in Z-ordering

`animate(framesPerSec);`
`animationPause();`
`animationResume();`
`animationStop();`
`isAnimated()`
	animation methods

`onAnimationTick()`	override for code to run on each anim. frame
`createFont(name, style)`	create a Typeface
`createPaint(red, green, blue)`	create a Paint

Types of GObjects

Class	Description
GColor	class with many Paint constants including BLACK, BLUE, RED, WHITE, etc.
GCompound	container for treating other objects as a group
GImage	represents a bitmap image
GLabel	a text string drawn in a given font
GLine	connection between two points
GObject	superclass for other graphical object classes
GOval	a circle or ellipse
GPolygon	connects arbitrary points to form a polygon
GRect	a square or rectangle
GSprite	wraps a GObject and adds methods useful for games

For details on each type of GObject, visit
http://web.stanford.edu/class/cs193a/lib/javadoc/

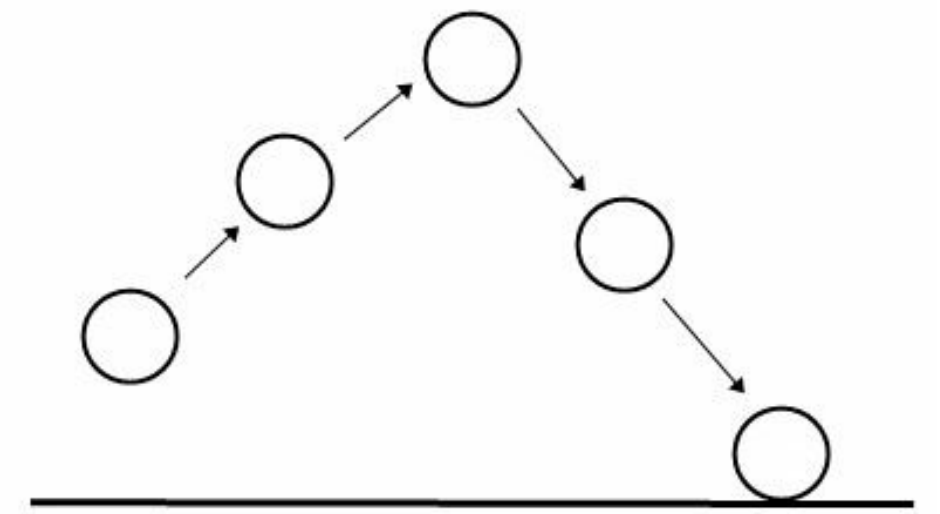
Smiley face with GCanvas

```
public class FaceView extends GCanvas { ...
        public void init() {
        GOval face = new GOval(100, 100, 200, 200); // face
        face.setColor(GColor.BLACK);
        face.setFillColor(GColor.YELLOW);
        add(face);

        GOval eye1 = new GOval(140, 140, 30, 30); // eyes
        eye1.setFillColor(GColor.BLUE);
        GOval eye2 = new GOval(230, 140, 30, 30);
        eye2.setFillColor(GColor.BLUE);
        add(eye1);
        add(eye2);

        GOval nose = new GOval(190, 180, 20, 20); // nose
        nose.setFillColor(GColor.BLACK);
        add(nose);

        GRect mouth = new GRect(170, 230, 60, 20); // mouth
        mouth.setFillColor(GColor.RED);
        add(mouth);

        GLabel label = new GLabel("Android is awesome", 100, 400);
        label.setFont(Typeface.MONOSPACE, Typeface.BOLD, 40f);    add(label);
        }
}
```

Animation via redrawing

- To animate a view, you must **redraw it** at regular intervals.
  - On each redraw, change variables/positions of shapes.

- Force a view to redraw itself by calling its `postInvalidate` method.
  - But you can't just do this in a loop; this will lock up the app's UI and lead to poor performance.
  - You must instead do it in another thread of execution.



A basic animation loop

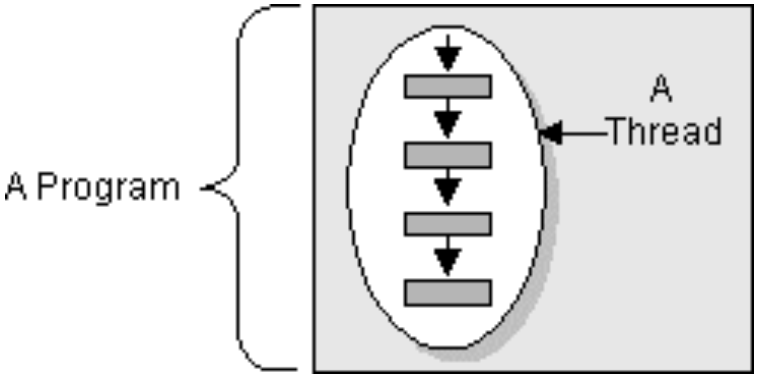
The code to animate a view must do the following in a loop:

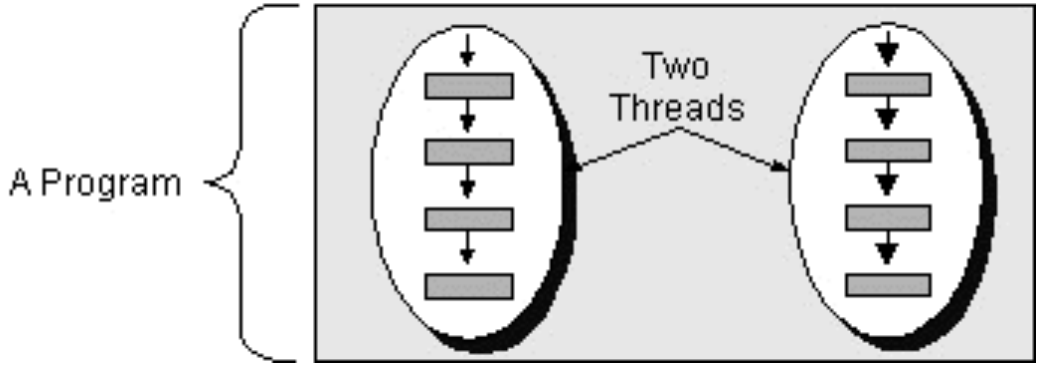
1. process any **user input** (mouse touch events, key presses, etc.)
2. **update** the view state (move any moving objects, handle collisions, etc.)
3. tell the view to **redraw** itself (which happens on the main UI thread)
4. **pause** for some number of milliseconds

```
// in MyView.java
public void myAnimationLoop() {
    while (true) {
            // 1) process user input
            // 2) update your game's state
            my game update code goes here;

            // 3) tell view to redraw self on main UI thread
            postInvalidate();

            // 4) pause
            try {
                    Thread.sleep(50); // 50ms = 20fps
            } catch (InterruptedException ie) { break; }
    }
}
```

Threads

- **thread:** A "lightweight process"; a single sequential flow of execution or isolated sub-task within one program.
  - A means to implement programs that seem to perform multiple tasks simultaneously (a.k.a. *concurrency*).
  - Threads within the same process share data with each other.
    - i.e., Variables created in one thread can be seen by others.
    - "shared-memory concurrency"
  - sometimes called a *lightweight process*





Using a Thread

- You can create a `Thread` by passing it a `Runnable` object with a `run()` method containing the code to execute.
  - other Thread methods: start, stop, sleep, isRunning, join

```
Thread thread = new Thread(new Runnable() {
    public void run() {
        // code to execute in thread goes here

    }
});
thread.start();
```

Redrawing a View in a Thread

- You can't just create a Thread and then call invalidate on your View from that thread.
  - Instead, you must use a "Handler" object to make the call, which requires its own second Runnable to do so.

```
// repaint the view a single time, in another thread
Thread thread = new Thread(new Runnable() {
    public void run() {
        Handler h = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
            public void run() {
                myView.invalidate();
            }
        });
    }
});
thread.start();
```

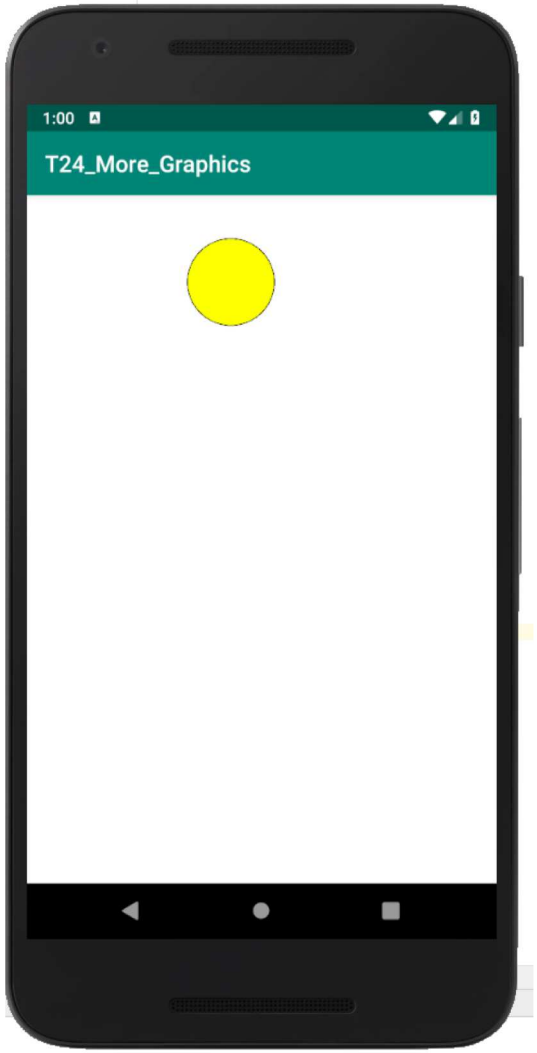
Avoid threads with library

- GCanvas includes an animate method that will call an onAnimateTick callback at specified intervals.
  - runs in a separate thread under-the-hood

```
// library eliminates threads, auto-redraws after each frame
public class MyView extends GCanvas { ...
        ...
        public void init() {
                animate(FPS); // frames/sec
        }

        // called once per frame of animation
        @Override
        public void onAnimateTick() {
                super.onAnimateTick();
                move/update shapes;
        }
}
```

Bouncing ball (x only) with library

```
// library eliminates threads, auto-redraws after each frame

public class BounceView extends GCanvas { ...
        private GOval ball;
        private int dx = 5;

        public void init() {
        ball = new GOval(10, 10, 50, 50);
        ball.setFillColor(GColor.BLACK);
        add(ball);
        animate(50); // 50 frames/sec
        }

        // called once per frame of animation
        @Override
        public void onAnimateTick() {
        super.onAnimateTick();
        ball.setX(ball.getX() + dx);
                if (ball.getRightX() >= getWidth()
                  || ball.getX() <= 0) {
                        dx = -dx; // bounce
                }
        }
}
```



A Sprite class

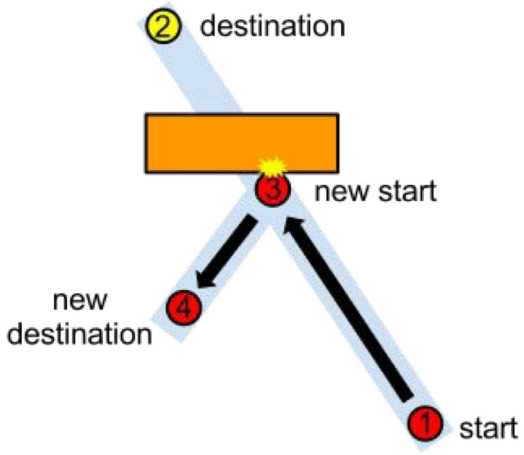
- **sprite**: An object of interest in a game.
  - possible data: location, size, velocity, shape/image, points, ...
  - Many games declare some kind of Sprite class to represent the sprites.

```
// an example sprite class
publicclass  Sprite {
    RectF rect;
    float dx, dy;
    Paint paint;
    ...
}
```

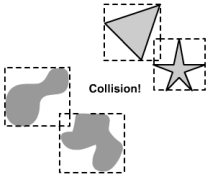
Collision detection

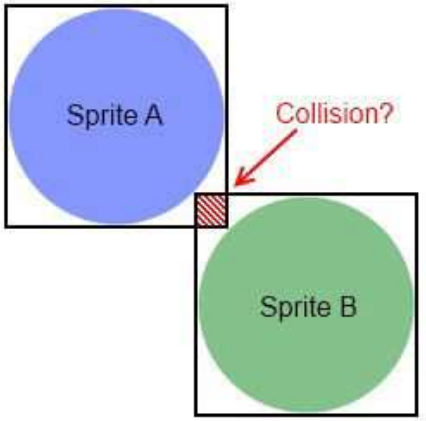
- **collision detection**: Determining whether sprites in the game world are touching each other (and reacting accordingly).

- Android's `RectF` (link) and other shapes have methods to check whether they touch:
  - `rect1.contains(x, y)`
  - `rect1.contains(rect2)`
  - `RectF.intersects(rect1, rect2)`

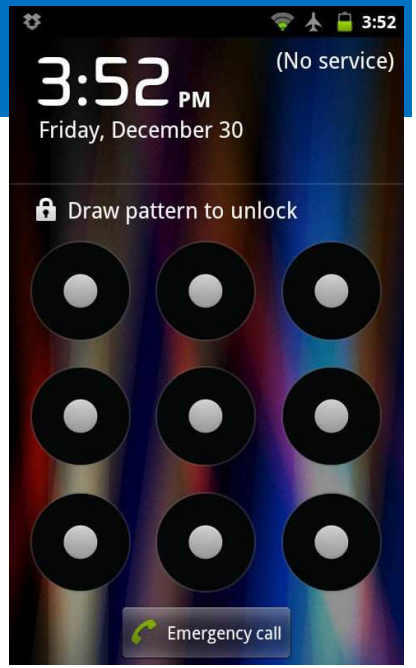


- Harder to compute for non-rectangular sprites.
- Some games use a smaller **collision rectangle** to give the collisions a bit of slack.





WakeLock



- To prevent screen from blanking, use a **wake lock**.
- in `AndroidManifest.xml`:

```
<uses-permission
 android:name="android.permission.WAKE_LOCK"  />
```

- in app's activity Java code:

```
// create the lock (probably in onCreate)
PowerManager pwr=(PowerManager)   getSystemService(POWER_SERVICE);
WakeLock lock=pwr.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
                              "my lock");

// turn on the lock (in onResume)
lock.acquire();

// turn off the lock (in onPause)
lock.release();
```

Full screen mode

- To put an app (e.g. a game) into full screen mode, which hides the notifications and status bar, put the following in your activity's onCreate method:

```
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(
        WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

Mouse touch events

- To handle finger presses from the user, write an onTouchEvent method in your custom View class.
  - actions: ACTION_DOWN, ACTION_UP, ACTION_MOVE, ...

```
@Override
public boolean onTouch(MotionEvent event)        {
    float x = event.getX();
    float y = event.getY();

    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        // code to run when finger is pressed


    }


    return super.onTouch(event);
}
```

Keyboard events

If you want to handle key presses (if the device has a keyboard):

- set your app to receive keyboard "focus" in View constructor:

```
requestFocus();
setFocusableInTouchMode(true);
```

- write onKeyDown/Up methods in your custom View class.
  - each key has a "code" such as `KeyEvent.KEYCODE_ENTER`

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_X) {
        // code to run when user presses the X key
    }
    return super.onKeyDown(keyCode, event);
}
```