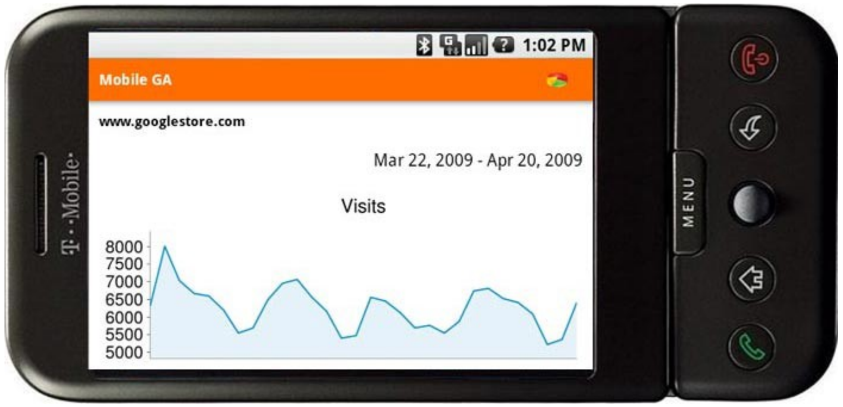
CS 193A

2D Graphics, Animation, and Games

Drawing 2D graphics

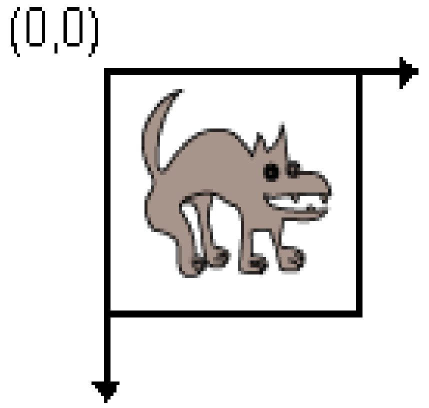
- To draw our own custom 2D graphics on screen, we'll make a **custom View subclass** with the drawing code.

- 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>
```

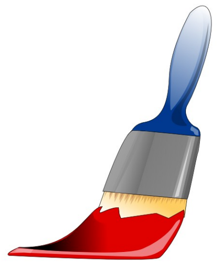
Canvas object methods (link)

- ***c***.drawARGB(***alpha***, ***r***, ***g***, ***b***); - fill window with color (rgb=0-255)
- ***c***.drawArc(...); - draw a partial ellipse
- ***c***.drawBitmap(***bmp***, ***x***, ***y***, null); - draw an image
- ***c***.drawCircle(***centerX***, ***centerY***, ***r***, ***paint***); - draw a circle
- ***c***.drawLine(***x1***, ***y1***, ***x2***, ***y2***, ***paint***); - draw a line segment
- ***c***.drawOval(***x1***, ***y1***, ***x2***, ***y2***, ***paint***); * *(requires Android 5.0)*
  c.drawOval(new RectF(***x1***, ***y1***, ***x2***, ***y2***), ***paint***); - draw oval/circle
- ***c.***drawPoint(***x***, ***y***, ***paint***); - color a single pixel
- ***c***.drawRect(***x1***, ***y1***, ***x2***, ***y2***, ***paint***); * *(requires Android 5.0)*
  c.drawRect(new RectF(***x1***, ***y1***, ***x2***, ***y2***), ***paint***); - draw rectangle
- ***c***.drawRoundRect(***x1***, ***y1***, ***x2***, ***y2***, ***rx***, ***ry***, ***paint***); * *(requires Android 5.0)*
  c.drawRoundRect(new RectF(***x1***, ***y1***, ***x2***, ***y2***), ***rx***, ***ry***, ***paint***);
- ***c***.drawText("***str***", ***x***, ***y***, ***paint***); - draw a text string
- ***c.***getWidth(), ***c***.getHeight() - get dimensions of drawing area

Paint (link)

- 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 has other useful methods like: getTextBounds, measureText, setAlpha, setAntiAlias, setStrokeWidth, setStyle, setTextAlign, setTextSize, setTypeface

Typeface (link)

- 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);
```

Bitmap images (link)

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

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

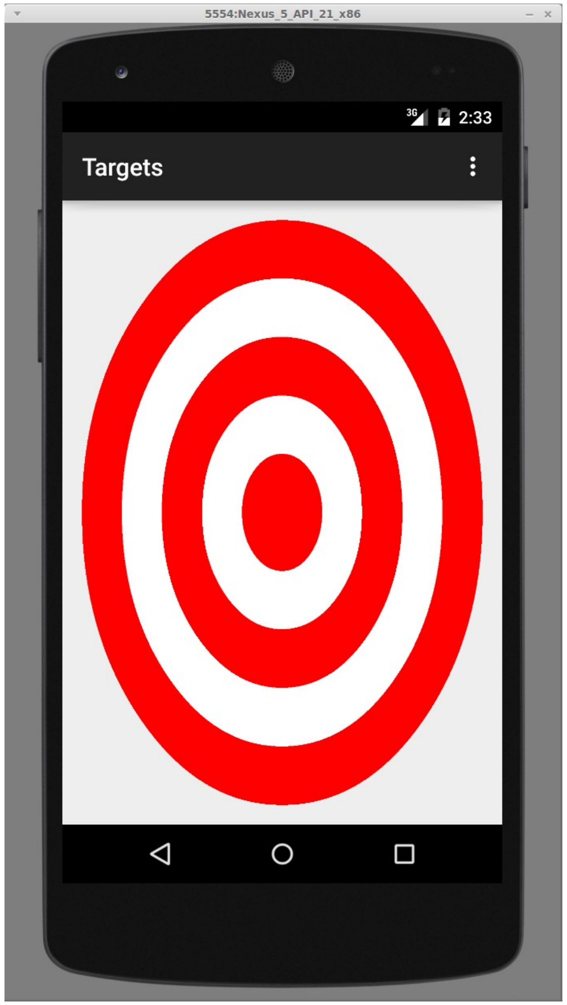
```
// example: draw heart.png on screen at (0, 0)
Bitmap bmp = BitmapFactory.decodeResource(
                 getResources(), R.drawable.heart);
canvas.drawBitmap(bmp, 0, 0, null);
```

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

Target exercise

- Write an app whose main activity displays a custom view that draws a "target" figure.
  - The outer red circle fills 100% of the main view's width and height.
  - There are 5 total circles, all centered; 3 red, 2 white.
  - Each circle is 20% smaller than the last:
    - the first (red) is 100% of the window size,
    - the second (white) is 80% of the window size,
    - the third (red) is 60% of the window size,
    - the fourth (white) is 40% of the window size,
    - the fifth (white) is 20% of the window size.

(Challenge: Can you introduce a constant so that the number of ovals is easy to change?)



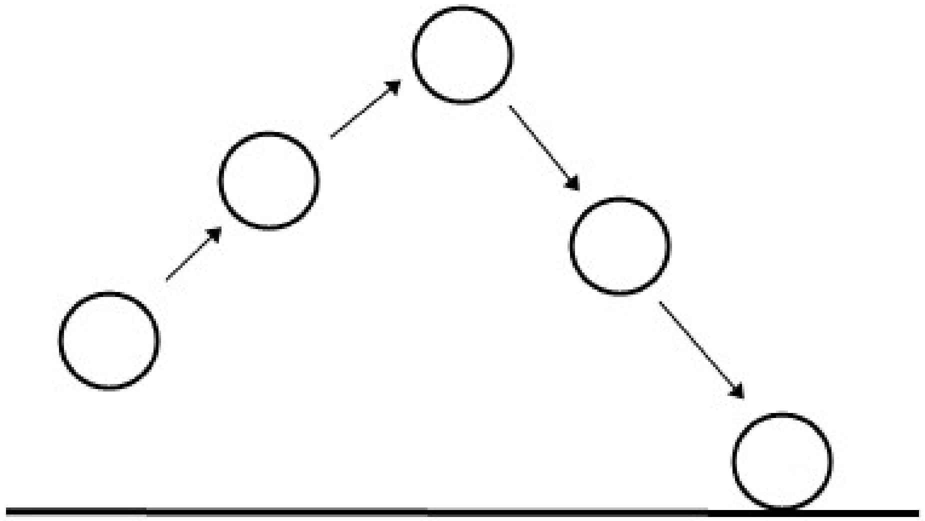
Target solution

```java
public class TargetView extends View {
    public TargetView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        Paint red = new Paint();
        red.setARGB(255, 255, 0, 0);
        Paint white = new Paint();
        white.setARGB(255, 255, 255, 255);

        int w = canvas.getWidth(), h = canvas.getHeight();
        for (int i = 0; i < 5; i++) {
            canvas.drawOval(new RectF(/*x*/ w*i/10,       /*y*/ h*i/10,
                                      /*w*/ w*(10-i)/10,  /*h*/ h*(10-i)/10),
                                 /*paint*/ i % 2 == 0 ? red : white);
        }
    }
}
```

Animation via redrawing

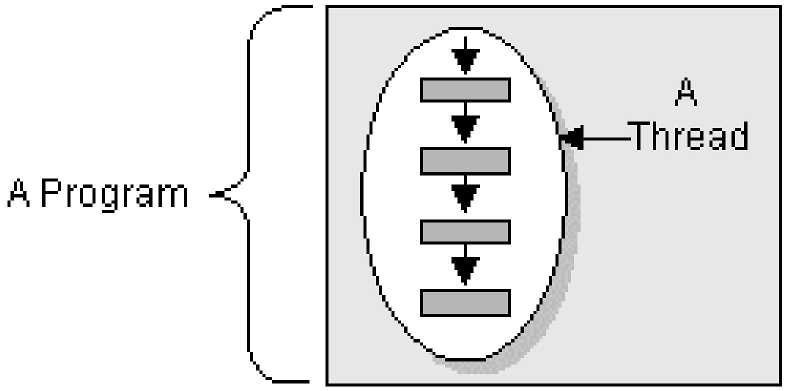
- To animate a view, you must **redraw it** at regular intervals.
  - On each redraw, change variables/positions of shapes.

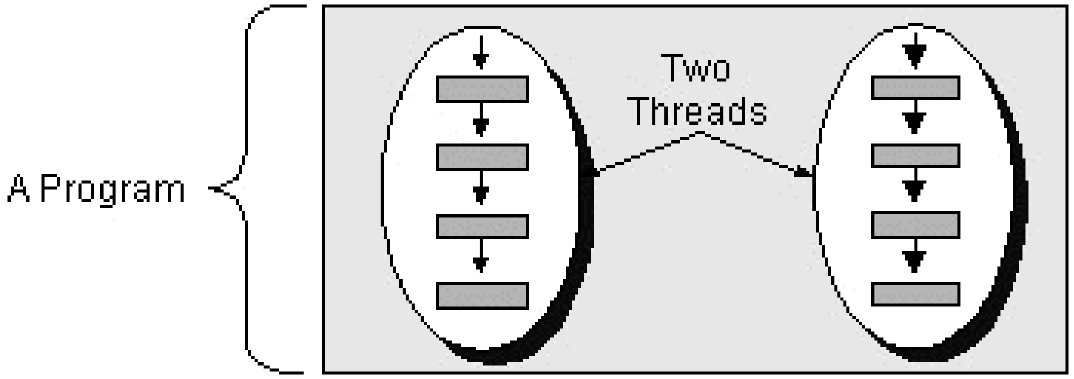
- Force a view to redraw itself by calling its `invalidate` method.
  - But you can't just do this in a loop; this will lock up the app's UI and lead to poor performance.



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

- Because of Android quirks, 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. (blargh!)

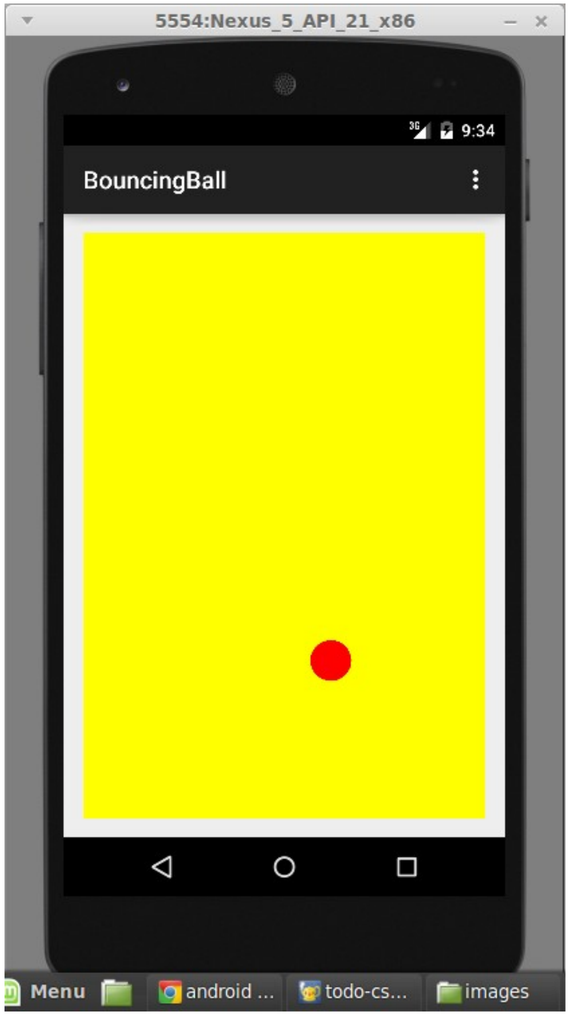
```
// repaint the view a single time, in another thread
Thread thread = new Thread(new Runnable() {
    public void run() {
        Handler h = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
            public void run() {
                myView.invalidate();
           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

Helper class: DrawingThread

- Because animation and threads are kind of icky, the instructor provides you a helper class named `DrawingThread`.
  - `public` **`DrawingThread`**`(`***`view`***`, `***`fps`***`)`
    Constructs a thread to redraw the given view the given number of times per second. *(Doesn't start it yet.)*
  - `public void` **`start`**`()`
    Starts the thread running.
  - `public void` **`stop`**`()`
    Halts the thread so it won't redraw any more.

Bouncing ball exercise

- Write an app that draws a bouncing red ball.
  The ball moves in the x/y dimensions and bounces back when it hits any edge of the screen.
  - background color: yellow
  - ball color: red
  - ball size: 100 x 100px
  - ball velocity: < 80px per in x/y direction (random)
  - ball should update 50 times per second



Mouse touch events (link)

- 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 onTouchEvent(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.onTouchEvent(event);
}
```

Keyboard events (link)

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);
```

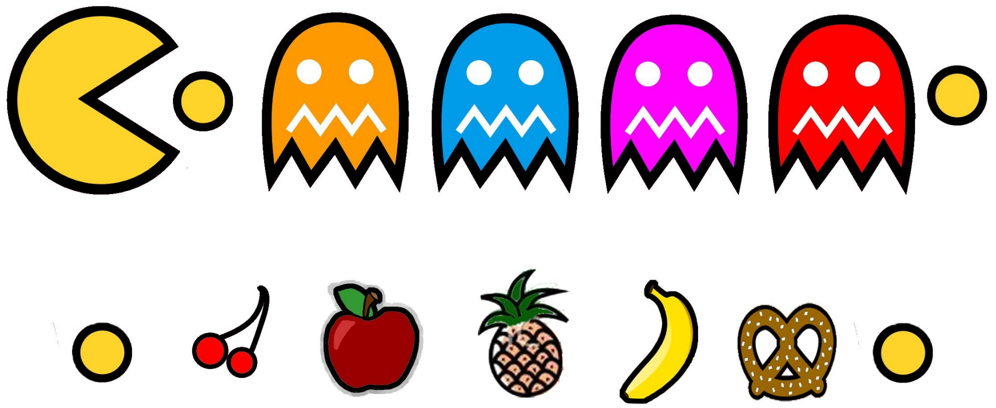
- write onKeyDown/Up methods in your custom View class.
  - each key has a "code" such as `KeyEvent.KEYCODE_ENTER`

```
@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);
}
```

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.

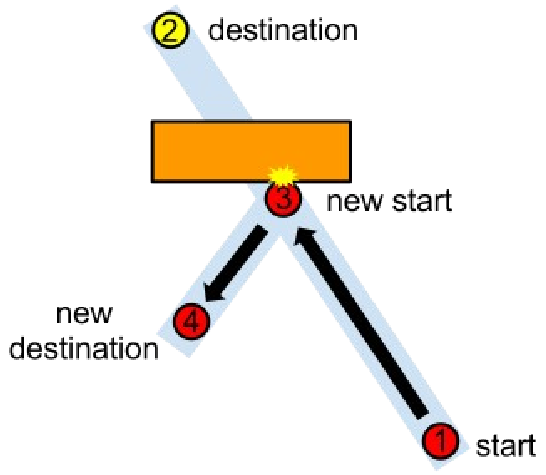
```
// an example sprite class
public class Sprite {
    RectF rect;
    float dx, dy;
    Paint paint;
    ...
}
```



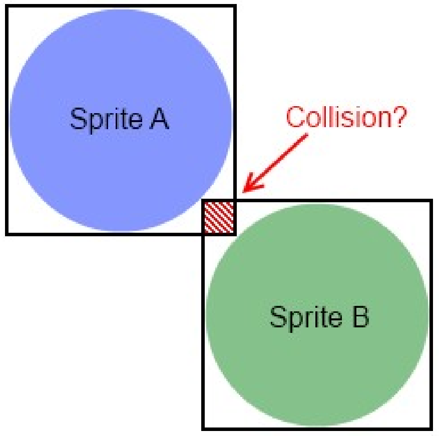
Collision detection

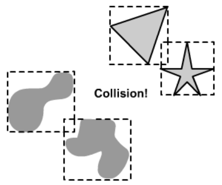
- **collision detection**: Determining whether sprites in the game world are touching each other (and reacting accordingly).

- 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*)`

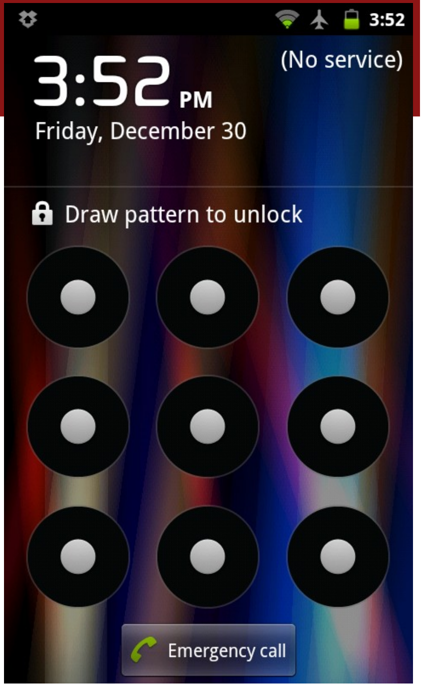


- Harder to compute for non-rectangular sprites.
- Some games use a smaller **collision rectangle** to give the collisions a bit of slack.





WakeLock



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

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

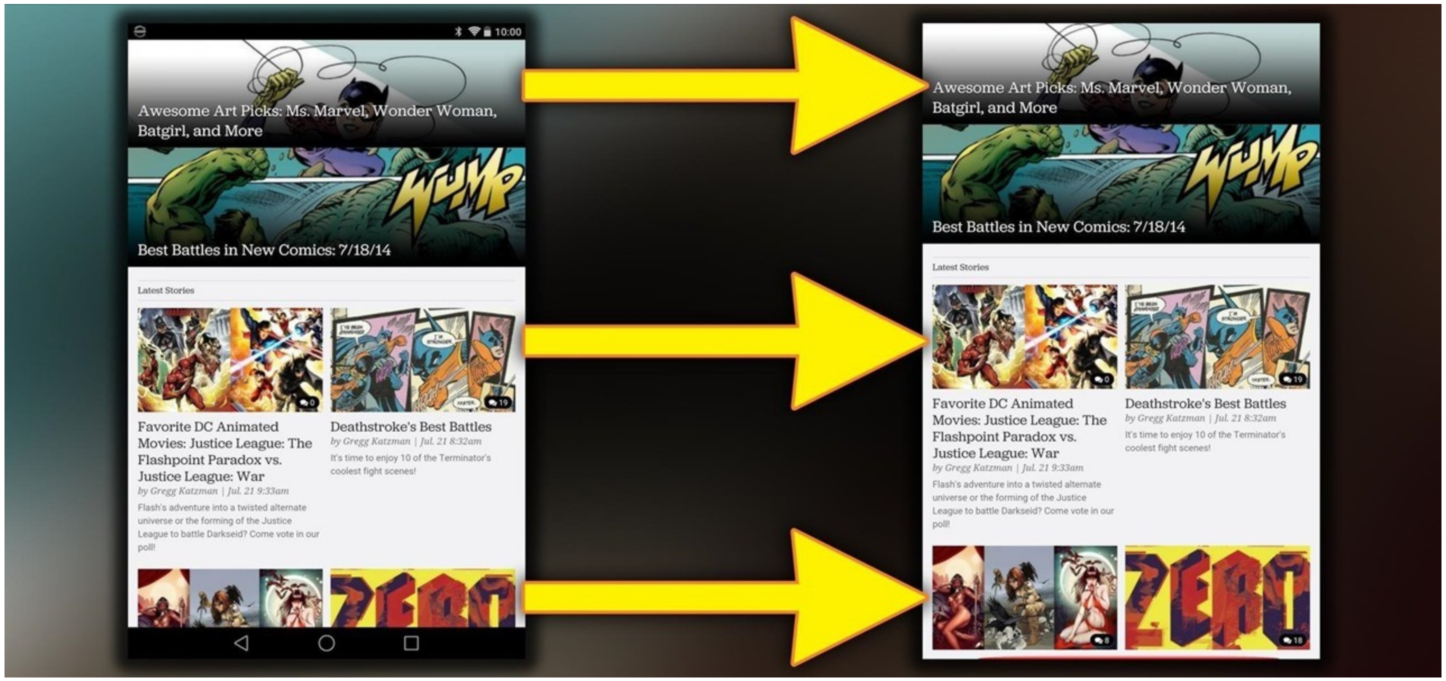
- in app's activity Java code:

```
// 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);
```


10:00
Awesome Art Picks: Ms. Marvel, Wonder Woman, Batgirl, and More
Best Battles in New Comics: 7/18/14
Latest Stories
Favorite DC Animated Movies: Justice League: The Flashpoint Paradox vs. Justice League: War
by Gregg Katzman | Jul. 21 9:33am
Flash's adventure into a twisted alternate universe or the forming of the Justice League to battle Darkseid? Come vote in our poll!
Deathstroke's Best Battles
by Gregg Katzman | Jul. 21 8:32am
It's time to enjoy 10 of the Terminator's coolest fight scenes!