Mobile Application (Design and) Development

7th class

Prof. Stephen Intille
s.intille@neu.edu

Q&A

- Workspace setup in lab. Anyone try it?
- Anyone looking for a partner?
- Boggle – various strategies
  - Buttons, swiping, 2d graphics, pause button, etc.

Today

- Overview of 2d graphics (part 1)
- Looking at graphics in Sudoku code

- Two design papers
  - Pixel Perfect Code: How to Marry Interaction and Visual Design the Android Way
    Presenter: Kevin Rottman
  - FIRST Else smartphone hands-on demo from Mobile World Congress 2010
    Presenter: Trevor Sontag

Schedule

- **Sunday: programming assignment 2 due**
- Monday: guest lecture Jason Nawyn
- Tuesday: in-class paper prototyping (bring what you need)
- Wednesday: guest lecture Fahd Albinali **Preliminary paper prototype design due**
- Thursday: location and sensing

- **Sunday: Programming assignment 3 due**

Graphics in Android

- Custom 2D graphics library
- OpenGL ES 1.0 for high performance 3D graphics
- Most common 2d graphics: android.graphics.drawable
- From ADG: "*When starting a project, it's important to consider exactly what your graphical demands will be*"

2d graphics option #1

- Draw into a View object from your layout
  - Drawing/animation of graphics handled by system's normal View hierarchy drawing process
  - You simply define graphics to go inside View
  - Call invalidate(); handle onDraw() callback
  - Best for drawing graphics that are not part of a performance-intensive game
  - Good for displaying static graphics or displaying animations within an otherwise static app

2d graphics option #2

- Draw your graphics directly to a Canvas
  - You call the appropriate class's draw() method (passing it your Canvas), or one of the Canvas draw...() methods
  - You are in control of animation
  - Best if you application needs to regularly redraw itself
  - Fast animation games must use this (or 3d)
  - Can use separate thread, manage a SurfaceView, and perform draws to the Canvas as fast as your thread is capable

2d graphics

- Drawable
  - Abstraction for “something that can be drawn”
  - Types
    - BitmapDrawable
    - ShapeDrawable
    - PictureDrawable
    - LayerDrawable
    - Others...

Drawable

- Three ways to define and instantiate:
  - Using an image saved in your project resources
  - Using an XML file that defines the Drawable properties
  - Using the normal class constructors

(E.g., my_image.png is referenced as *my_image*).

Creating from resources

- Easy
- Image types:
  - PNG (preferred)
  - JPG (acceptable)
  - GIF (discouraged)
- Add your file to the res/drawable/ directory
  - E.g., my_image.png is referenced as *my_image*

ImageView using resources

```
LinearLayout mLinearLayout;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Create a LinearLayout in which to add the ImageView
    mLinearLayout = new LinearLayout(this);

    // Instantiate an ImageView and define its properties
    ImageView i = new ImageView(this);
    i.setImageResource(R.drawable.my_image);
  // set the ImageView bounds to match the Drawable's dimensions
    i.setAdjustViewBounds(true
    i.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
                                         LayoutParams.WRAP_CONTENT));

    // Add the ImageView to the layout and set the layout as the content view
    mLinearLayout.addView(i);
    setContentView(mLinearLayout);
}
```

Handling image res as drawable

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```

Add a resource Drawable to an ImageView in the XML layout:

```
<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:tint="#55ff0000"
  android:src="@drawable/my_image"/>
```

Creating from resource XML

res/drawable/expand_collapse.xml:

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/image_expand">
  <item android:drawable="@drawable/image_collapse">
</transition>
```

```
Resources res = mContext.getResources();
TransitionDrawable transition = (TransitionDrawable)
res.getDrawable(R.drawable.expand_collapse);
ImageView image = (ImageView) findViewById(R.id.toggle_image);
image.setImageDrawable(transition);
```

Run forward for 1s:

```
transition.startTransition(1000);
```

ShapeDrawable

```
public class CustomDrawableView extends View {
    private ShapeDrawable mDrawable;

    public CustomDrawableView(Context context) {
        super(context);

        int x = 10;
        int y = 10;
        int width = 300;
        int height = 50;

        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(0xff74AC23);
        mDrawable.setBounds(x, y, x + width, y + height);
    }

    protected void onDraw(Canvas canvas) {
        mDrawable.draw(canvas);
    }
}
```

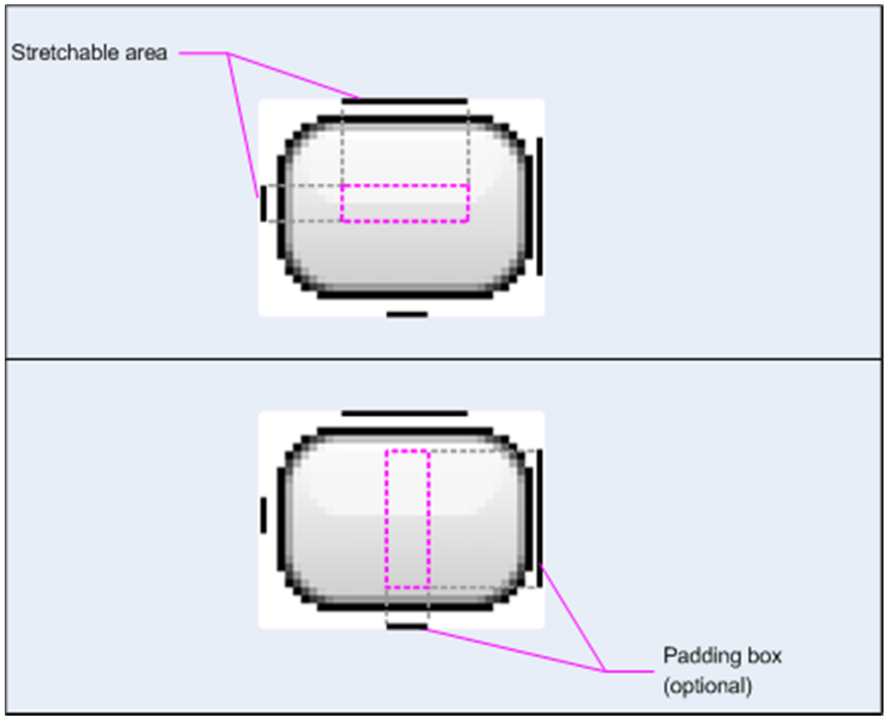
ShapeDrawable

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCustomDrawableView = new CustomDrawableView(this);

    setContentView(mCustomDrawableView);
}
```

Nine-patch

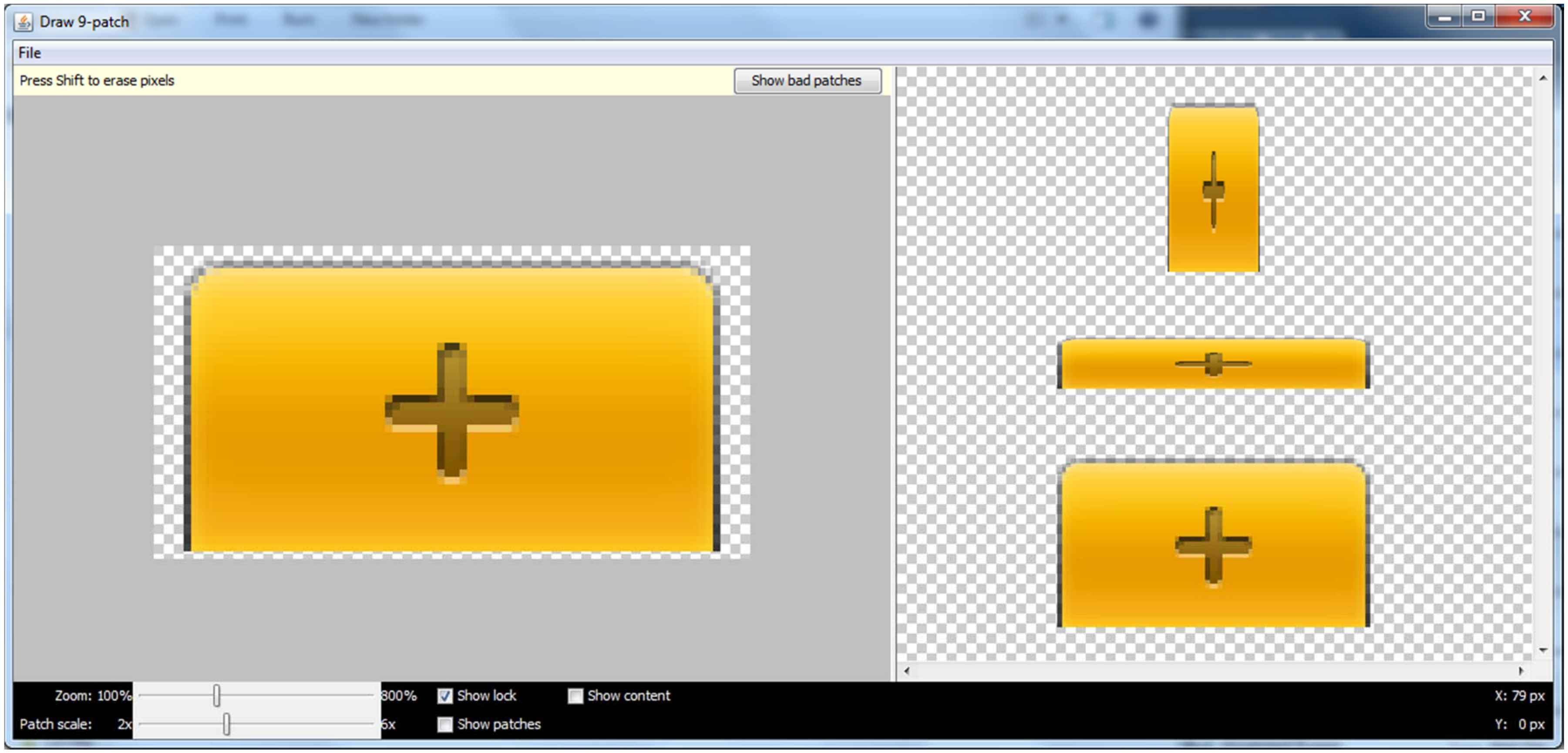


- Top image shows pixels that will be replicated for stretching

- Bottom image identifies the region in which the contents of the View are allowed. If the contents don't fit in this region, then the image will be stretched so that they do.

Draw nine-patch tool

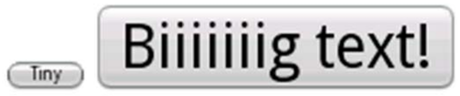
- VERY useful



Adding drawable to button

```
<Button id="@+id/tiny"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerInParent="true"
        android:text="Tiny"
        android:textSize="8sp"
        android:background="@drawable/my_button_background"/>

<Button id="@+id/big"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerInParent="true"
        android:text="Biiiiiiig text!"
        android:textSize="30sp"
        android:background="@drawable/my_button_background"/>
```



What is a canvas?

- Interface to drawing surface
- Holds "draw" calls
- Actual drawing uses a Bitmap put onto the window
- You can...
  - Use OnDraw() (Canvas provided)
  - Get canvas (using SurfaceView objects)
  - Create new canvas...

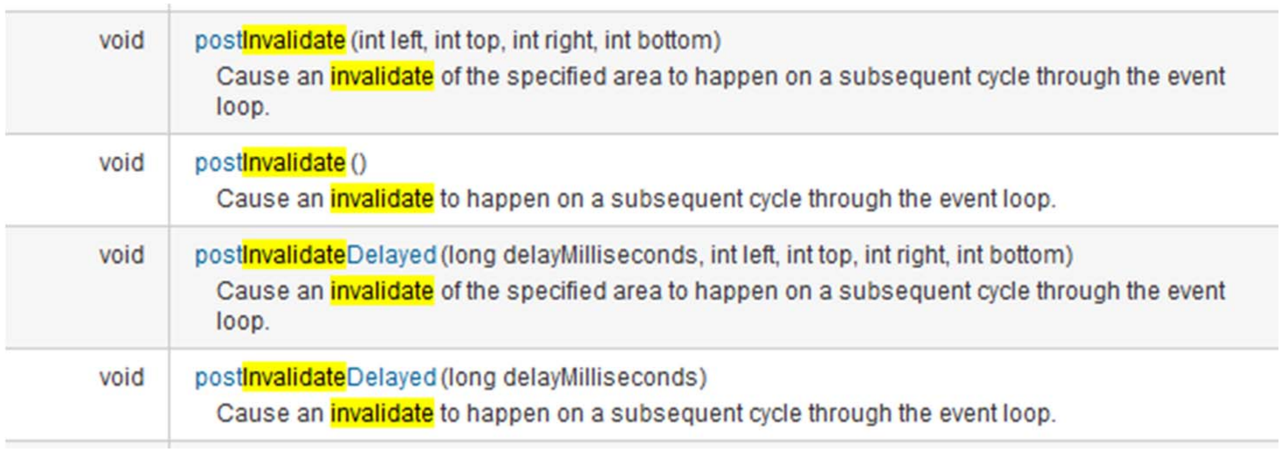
Create new canvas

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

- You can then carry to another Canvas using a Canvas.drawBitmap(Bitmap,...) method
- Best to draw final graphics via Canvas from View.onDraw() or SurfaceHolder.lockCanvas()

Using View.onDraw()

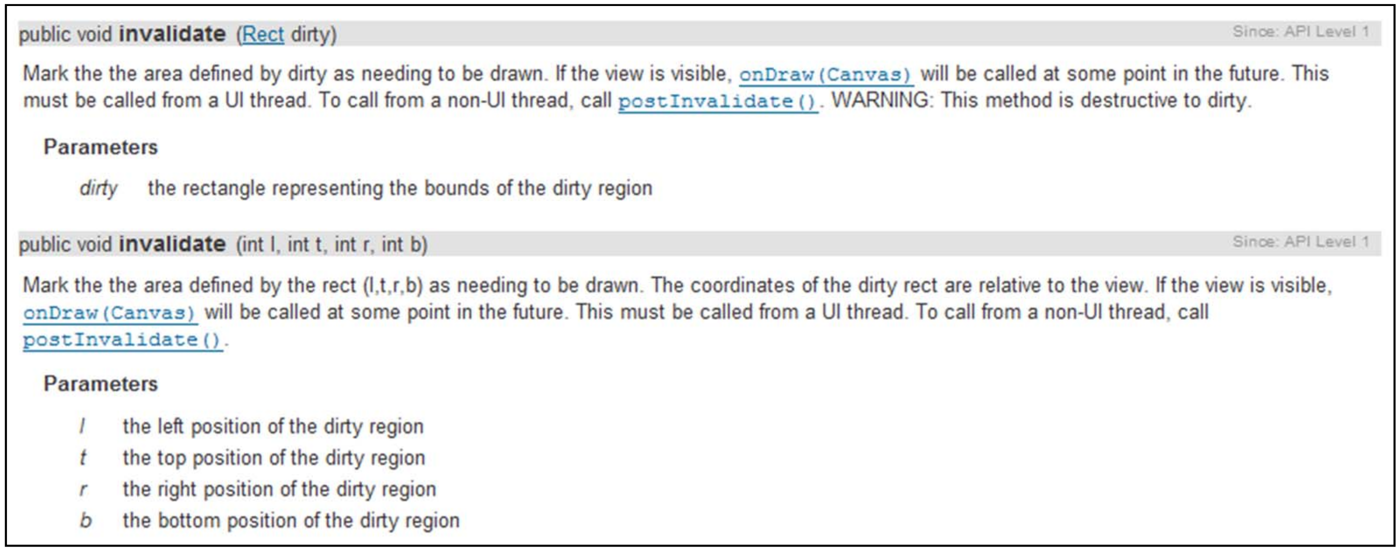
- Android only calls onDraw() as necessary
- Use invalidate() to indicate a redraw is necessary
  - Better:



void	postInvalidate (int left, int top, int right, int bottom) Cause an invalidate of the specified area to happen on a subsequent cycle through the event loop.
void	postInvalidate () Cause an invalidate to happen on a subsequent cycle through the event loop.
void	postInvalidateDelayed (long delayMilliseconds, int left, int top, int right, int bottom) Cause an invalidate of the specified area to happen on a subsequent cycle through the event loop.
void	postInvalidateDelayed (long delayMilliseconds) Cause an invalidate to happen on a subsequent cycle through the event loop.

Using View.onDraw()

- Android only calls onDraw() as necessary
- Use invalidate() to indicate a redraw is necessary
  - Not immediate
  - Better:



public void **invalidate** (Rect dirty) Since: API Level 1

Mark the the area defined by dirty as needing to be drawn. If the view is visible, `onDraw(Canvas)` will be called at some point in the future. This must be called from a UI thread. To call from a non-UI thread, call `postInvalidate()`. WARNING: This method is destructive to dirty.

Parameters

dirty	the rectangle representing the bounds of the dirty region

public void **invalidate** (int l, int t, int r, int b) Since: API Level 1

Mark the the area defined by the rect (l,t,r,b) as needing to be drawn. The coordinates of the dirty rect are relative to the view. If the view is visible, `onDraw(Canvas)` will be called at some point in the future. This must be called from a UI thread. To call from a non-UI thread, call `postInvalidate()`.

Parameters

l	the left position of the dirty region
t	the top position of the dirty region
r	the right position of the dirty region
b	the bottom position of the dirty region

Using View.onDraw()

- Use Canvas and all Canvas.draw...() methods
- After onDraw() finishes, Android does the work to draw actual Bitmap

(Snake example in API)

Using SurfaceView

- Dedicated drawing surface within the View hierarchy
- Offer a drawing surface to a secondary thread (decouple from UI/View hierarchy)

(Lunar Lander example in API)

Sudoku

- Looking at/questions about code...

Readings