

Mobila applikationer och trådlösa nät, HT2014

HI1033

Lecturer: Anders Lindström,
anders.lindstrom@sth.kth.se

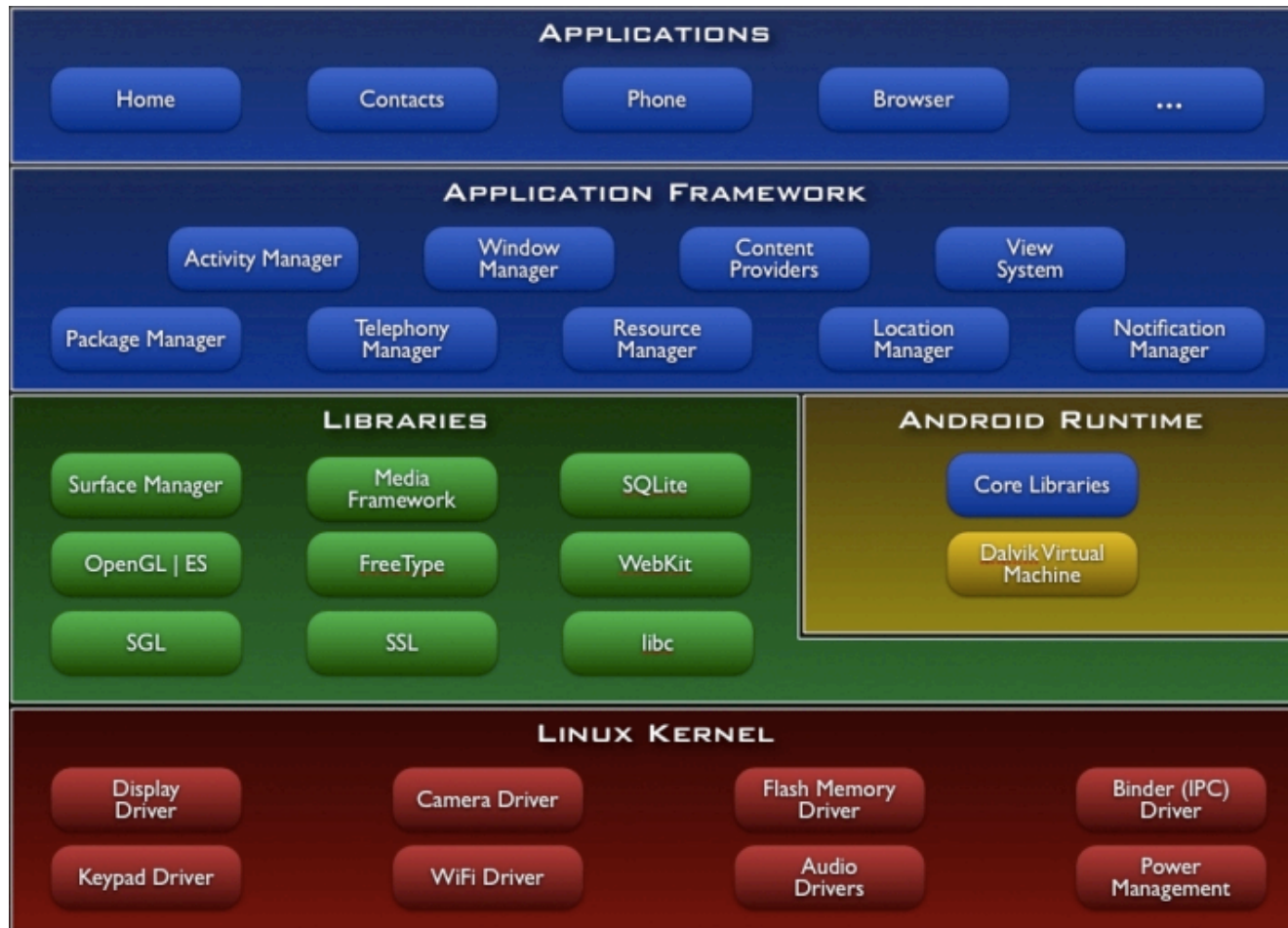
Lecture 6

Today's topics

- Android graphics
 - Views, Canvas, Drawables, Paint
 - Double buffering, bitmaps
 - Animations
 - Graphics and multi threading, SurfaceView



Android graphics



Android graphics

- 2D graphics library, package android.graphics, android.view
 - Android 4.0: Hardware accelerated rendering pipeline
- OpenGL ES 1.0, 2D and 3D, hardware acceleration, package android.opengl
- Issues:
 - Screens with different sizes and resolutions
 - Performance

Android 2D graphics

3 different needs, 3 different ways to draw

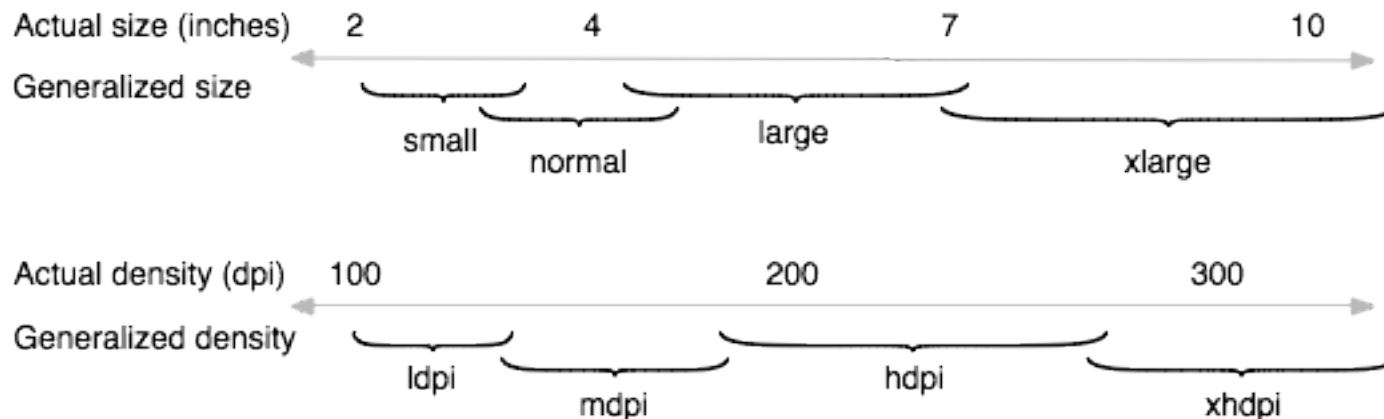
1. "Static" – define views, drawings and animations in layout files
2. Update views when needed, e.g. on user input. Draw on a Canvas (`View.onDraw`), called implicitly via `View.invalidate` (from main-thread)
 - 1 and 2 may include Animation-objects
3. In a separate thread, wherein you manage a `SurfaceView` and perform painting directly to a Canvas

Drawables

- `android.graphics.drawable`
- Something that can be drawn, e.g. an image, shape, transition, animation
- Create Drawables from resources, xml layout or by calling a constructor
- Images are stored in `res/drawable-ldpi`, `/...-mdpi`, `/...-hdpi`
- Preferred format: `png`, `9.png` (acceptable: `jpg`)

Drawables

- Screen size
- Screen density
- Orientation
- Resolution (pixels) or density
 - not the same as dp, density independent pixel (used in application code)



Source: http://developer.android.com/guide/practices/screens_support.html

Drawables, from layout file

- Define a view with an image in layout xml:

```
<ImageView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:src="@drawable/my_image"/>
```

- Create a drawable object (image) from resources:

```
Resources res = mContext.getResources();
```

```
Drawable myImage =
```

```
    res.getDrawable(R.drawable.my_image);
```

Drawables, in code

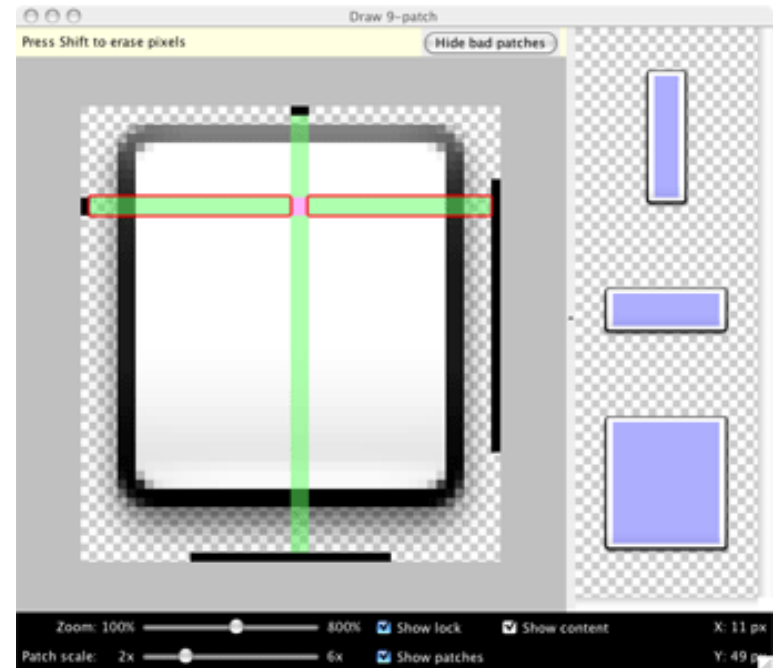
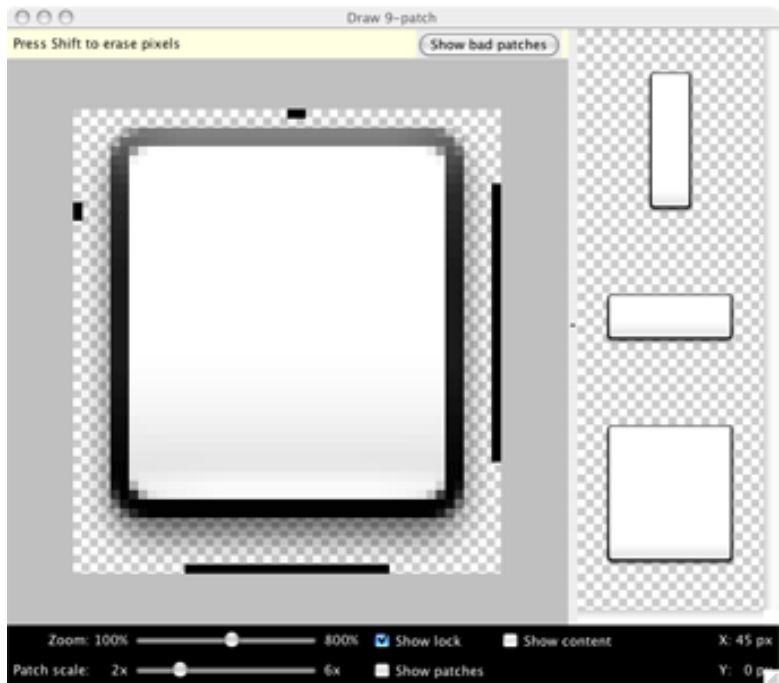
- Load image into ImageView in code:

```
protected void onCreate(Bundle savedInstanceState) {  
    . . .  
    LinearLayout layout = new LinearLayout(this);  
  
    ImageView iw = new ImageView(this);  
    iw.setImageResource(R.drawable.my_image);  
    iw.setAdjustViewBounds(true);  
    . . .  
    layout.addView(iw);  
    setContentView(layout);  
    . . .  
}
```

- In this case: do not call `setContentView(R.layout.main)`

Drawables, 9-patch stretchable

- Stretchable png-image (9.png)



- Editor in SDK: `tools/draw9patch.bat`

Drawables, 9-patch stretchable

```
<Button
```

```
    ..  
    id="@+id/tiny"  
    android:text="Tiny"  
    android:textSize="8sp"  
    android:background="@drawable/my_button_background"
```

```
/>
```

```
<Button id="@+id/big"
```

```
    ..  
    android:text="Biiiiiiig text!"  
    android:textSize="30sp"  
    android:background="@drawable/my_button_background"
```

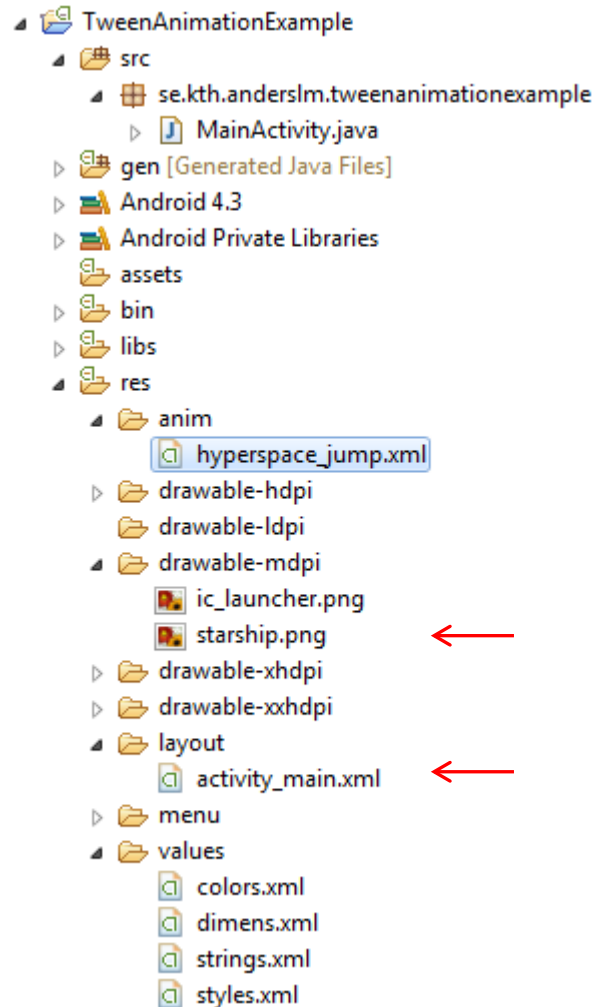
```
/>
```



(be)Tween Animation

- Package `android.view.animation`
- Simple transformations: translation, rotation, size, transparency (alpha)
- Define the transformations that you want to occur, when they will occur, and how long they should take to apply
- Transformations can be sequential or simultaneous
- Define in `res/anim/my_animation.xml`

Tween Animation



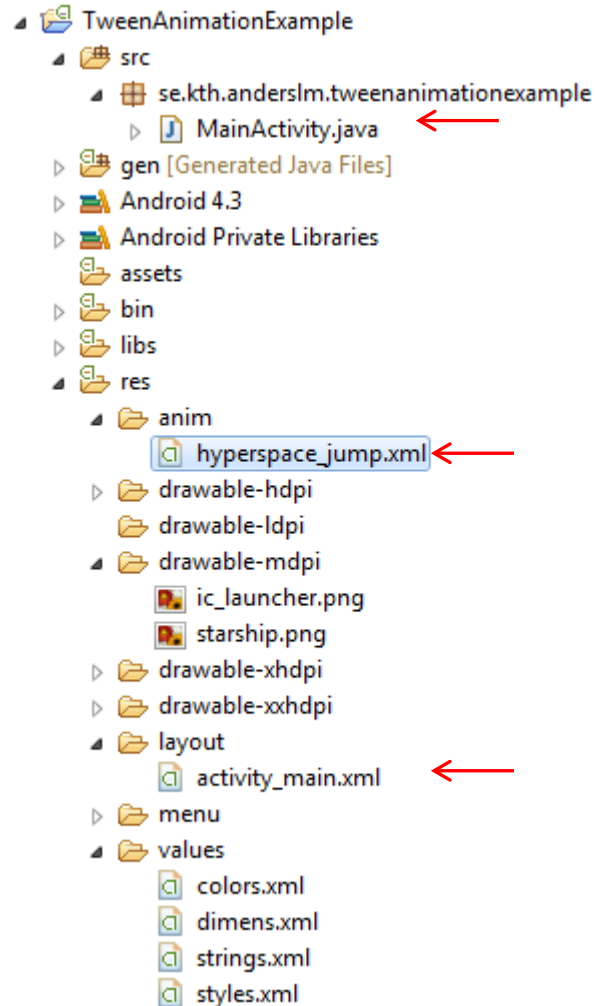
I layoutfilen:

<ImageView

```
android:id="@+id/HyperspaceImageView"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:contentDescription=  
    "@string/starship"  
android:src="@drawable/starship"
```

/>

Tween Animation



```
spaceshipImage = (ImageView)
    findViewById(R.id.HyperspaceImageView);
hyperspaceJumpAnimation =
    AnimationUtils.loadAnimation(
        this, R.anim.hyperspace_jump);
```

...

```
spaceshipImage.startAnimation(
    hyperspaceJumpAnimation);
```

Tween Animation

- res/anim/hyperspace_jump.xml (not complete, see code ex.)

```
<set ... >
  <scale
    android:interpolator="@android:anim/linear_interpolator"
    android:duration="700"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:toXScale="1.4"
    android:toYScale="0.6" />

  <set android:interpolator="@android:anim/decelerate_interpolator"
    android:duration="2000"
    android:startOffset="700">
    <scale
      android:fromXScale="1.4"
      android:fromYScale="0.6"
      ... />
    <rotate
      android:duration="2000"
      android:fromDegrees="0"
      android:toDegrees="360"
      android:pivotX="50%"
      ... />
  </set>
</set>
```

Frame Animation

- `res/anim/rocket_thrust.xml`
- ```
<animation-list
 ...
 android:oneshot="true">
 <item android:drawable="@drawable/rocket_thrust1"
 android:duration="200" />
 <item android:drawable="@drawable/rocket_thrust2"
 android:duration="200" />
 <item android:drawable="@drawable/rocket_thrust3"
 android:duration="200" />
</animation-list>
```

# Frame Animation

- Corresponding code:

```
private AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
 ...

 ImageView rocketView = (ImageView)
 findViewById(R.id.rocket_image);
 rocketView.setBackgroundResource(
 R.drawable.rocket_thrust);

 rocketAnimation = (AnimationDrawable)
 rocketView.getBackground();
}

public void onSomeClick(View v) {
 rocketAnimation.start();
}
```



# Update views from main-thread

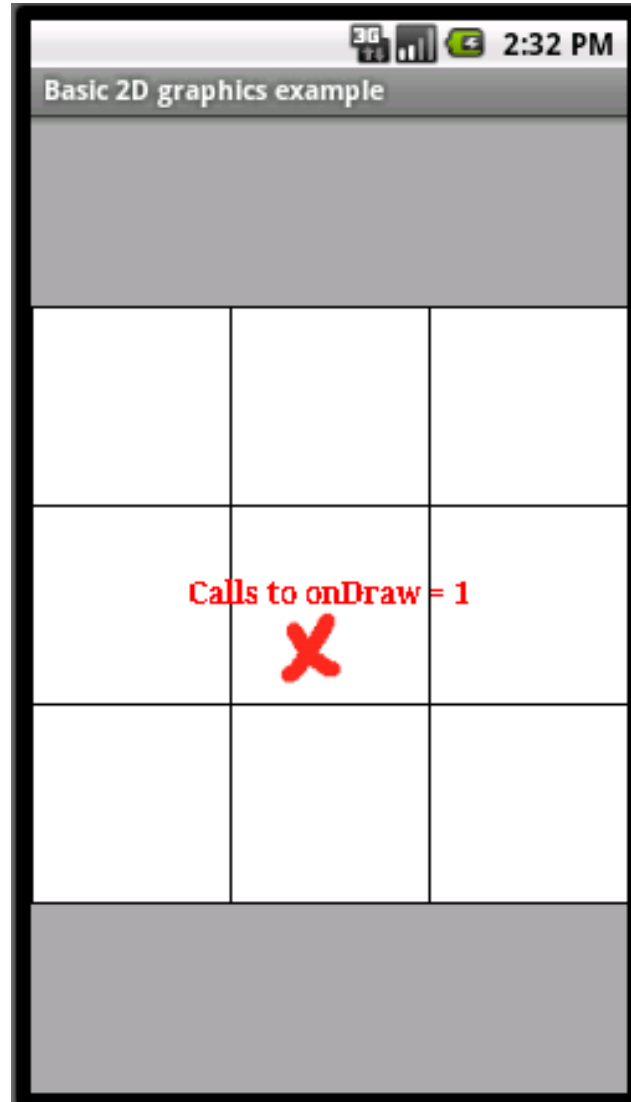
## Graphic components

- Canvas – defines what to "draw", holds a bit map representing the pixels to draw
- Drawables for drawing primitives, e.g. Rect, Path, text, Bitmap, image, Animations
- Paint - describes colors and styles when drawing
- Color (int), Typeface, ...
  
- Define what to draw by extending the View class and override `onDraw(Canvas)`

# Canvas

- Holds the surface upon which your graphics will be drawn and thus all of your "draw" calls
- `public class MyView extends View {`
  - `public void onDraw(Canvas canvas) {`
    - `Paint paint = new Paint();`
    - `paint.setColor(Color.WHITE);`
    - `canvas.drawPaint(paint);`
    - `• • •`
  - `}`
- `private void onSomeAction(...) {`
  - `this.invalidate(); // Request a call to onDraw`
  - `}`

# onDraw + Canvas



Ex: Basic2DGraphics

# onDraw + Canvas

```
public class BasicGraphicsView extends View {

 @Override
 protected void onDraw(Canvas canvas) {

 // Current size of this view
 int w = this.getWidth(), h = this.getHeight();
 int offset = (h - w)/2;

 // Background
 Paint bgPaint = new Paint();
 bgPaint.setColor(colorLightGrey);
 canvas.drawPaint(bgPaint);

 // Fill a rectangle
 Paint rectPaint = new Paint();
 rectPaint.setColor(Color.WHITE);
 canvas.drawRect(0, offset, w, h-offset, rectPaint);
 }
}
```

# onDraw, draw an image

```
public class BasicGraphicsView extends View {
 private Drawable cross; // An image representation
 public BasicGraphicsView(Context context) {
 super(context);
 Resources resources = context.getResources();
 cross = (Drawable)
 resources.getDrawable(R.drawable.cross);
 }
```

```
protected void onDraw(Canvas canvas) {
 int x = 100, y = 200;
 int iw = cross.getIntrinsicWidth();
 int ih = cross.getIntrinsicHeight();
 Rect bounds = new Rect(x, y, x+iw, y+ih);
 cross.setBounds(bounds);
 cross.draw(canvas);
}
```

# Canvas, off screen drawing

- Drawing is performed on an underlying Bitmap holding the pixels
- You can create a new Canvas, e.g. for off screen drawing
- Provide the Bitmap upon which drawing will actually be performed
- ```
Bitmap offscreen = Bitmap.createBitmap(
    width, height, Bitmap.Config.ARGB_8888);
```

```
Canvas temp = new Canvas(offscreen);
```

```
// Draw off screen, using the offscreen canvas
```

```
temp.drawRect(. . .);
```

```
. . .
```

```
// Copy the bitmap to the Canvas associated with screen
canvas.drawBitmap(offscreen, . . .);
```

SurfaceView

- Provides a surface on which a *secondary thread* may render into the screen at it's own chosen speed
- Surface – a handle to a raw buffer being managed by the screen composer
- Access to the surface is provided via the SurfaceHolder interface
 - retrieved by calling `getHolder()`
- Use SurfaceView when your view constantly needs updates, e.g. a game view
- Penalty: Memory consuming
- Implement SurfaceHolder.Callback – methods called (by the main-thread) when the surface is created, changed, or destroyed
 - the scondary thread must only touch the underlying Surface between `SurfaceHolder.Callback.surfaceCreated()` and `SurfaceHolder.Callback.surfaceDestroyed()`.

SurfaceView

```
public void run() {
    while(running) {
        . . .
        // TO DO: Draw on an off screen Bitmap before
        // calling holder.lockCanvas()

        Canvas canvas = holder.LockCanvas();
        {
            Paint paint = new Paint();
            paint.setColor(Color.WHITE);
            canvas.drawPaint(paint);
            . . .
        }
        holder.unlockCanvasAndPost(canvas);

        try {
            Thread.sleep(time);
        }
        ...
    }
}
```


SurfaceView

The Activity holding the SurfaceView and graphics thread must override the appropriate life cycle call-backs

```
public class SurfaceActivity extends Activity {
    private SnowSurfaceView view;

    public void onCreate(Bundle savedInstanceState) {
        . . .
        setContentView(view);
    }

    protected void onResume() {
        super.onResume();
        view.resume(); // stop drawing (thread)
    }

    protected void onPause() {
        super.onPause();
        view.pause(); // if hasSurface, start drawing (thread)
    }
}
```

SurfaceView + graphics thread

- Model tasks and objects in the game as classes
- Use SurfaceView + thread
- Off screen drawing
- Manage life cycle call backs
- Reuse objects (e.g. Paint)
- Check for actual screen size and changes in size and orientation



SurfaceViewExample.zip

Add a custom view to a layout file

```
public class CustomView extends View {
```

```
→ public CustomView (Context context, AttributeSet attrs) {  
    super(context, attrs);  
    ...  
}
```

```
public CustomView(Context context) {  
    this(context, null);  
}
```

```
@Override
```

```
protected void onDraw(Canvas canvas) {  
    int w = this.getWidth(), h = this.getHeight();  
    ...  
    canvas.drawCircle(cx, cy, radius, paint);  
}
```

```
...
```

Add a custom view to a layout file

```
<RelativeLayout
```

```
  ...>
```

```
  <TextView
```

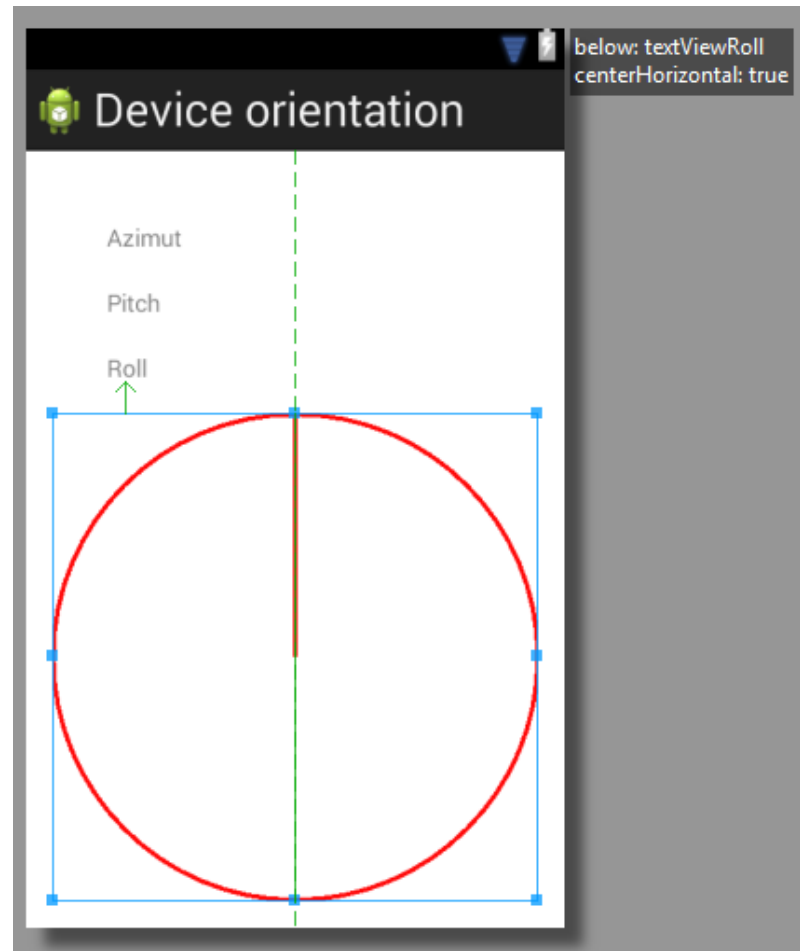
```
    .../>
```

```
  ...
```

```
→ <se.kth.anderslm.myapplication.CustomView  
    android:id="@+id/customView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:layout_below="@+id/textView1"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="17dp" />
```

```
</RelativeLayout>
```

Add a custom view to a layout file



Add a custom view to a layout file

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        theTextView1 = (TextView)  
        findViewById(R.id.textView1);  
        ...  
        → customView = (CustomView)  
        findViewById(R.id.customView);  
        ...  
    }  
}
```

Touch Events

- Fired when user touches the screen
- Listen to touch event:
Extend View and override onTouchEvent
(MotionEvent event) , or ...
- Implement View.OnTouchListener and override
onTouch(View v, MotionEvent event)

Touch Events

```
@Override
public boolean onTouchEvent(MotionEvent event) {

    int action = event.getAction();

    switch (action) {
    case (MotionEvent.ACTION_DOWN):
        // Touch screen pressed
        break;
    case (MotionEvent.ACTION_UP):
        // Touch screen touch ended
        break;
    case (MotionEvent.ACTION_MOVE):
        // Moved across screen
        break;
    case (MotionEvent.ACTION_CANCEL):
        // Touch event cancelled
        break;
    }
    return super.onTouchEvent(event);
}
```


Touch Events

```
public class TouchView extends View {
    . . .
    @Override
    public boolean onTouchEvent(MotionEvent event) {

        if(event.getAction() == MotionEvent.ACTION_DOWN) {
            int w = cross.getIntrinsicWidth();
            int h = cross.getIntrinsicHeight();
            int x = (int) event.getX();
            int y = (int) event.getY();
            crossBounds = new Rect(x-w/2, y-w/2, x+w/2, y+h/2);

            invalidate(); // Request a redraw of this view
            return true;
        }
        return false;
    }
}
```

Readings

- <http://developer.android.com/guide/topics/graphics/index.html>
- Code examples at Bilda
- Meier, Chapter 4 and 11

Bonus: Define a "listener callback" via a property

- The layout file

```
<Button
```

```
    ...  
    android:id="@+id/StartButton"  
    android:text="@string/start"  
    android:onClick="onStartClick" >
```

```
</Button>
```

- The Activity

```
public void onStartClick (View view) {
```

```
    ...
```

```
}
```

- Considered bad practice(?)