Lecture 11:
2D Graphics with Views

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Making your own View

Draw views using the `View.onDraw()` Function

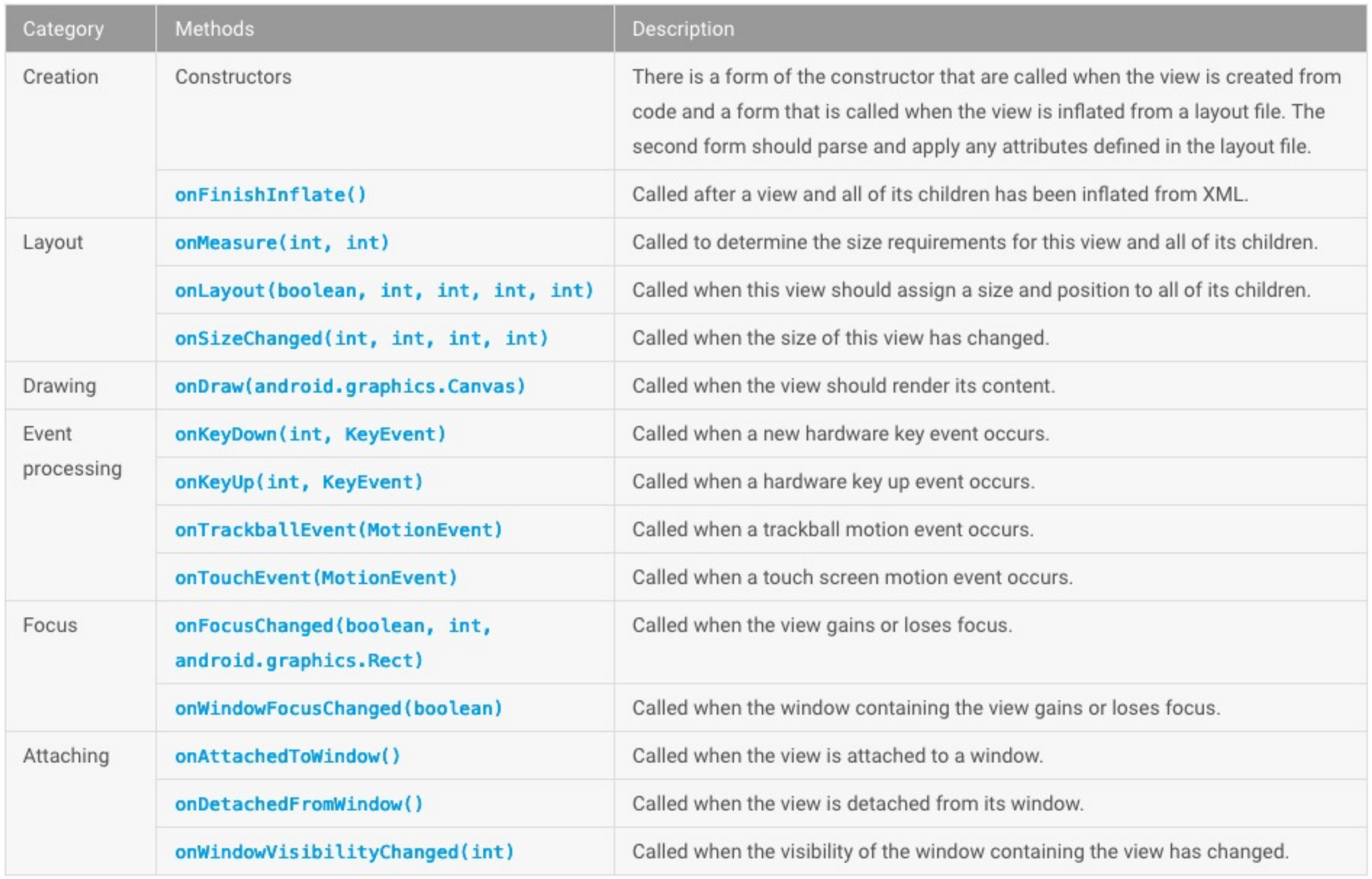
(touch events will be taught later)

Draw on Views

- To draw on customised `View` objects, you need to override its `onDraw()` method.
  - It is called by the system whenever the view needs to be refreshed.

```
class MyView extends View {
    …

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

Other Methods (Passively) Called

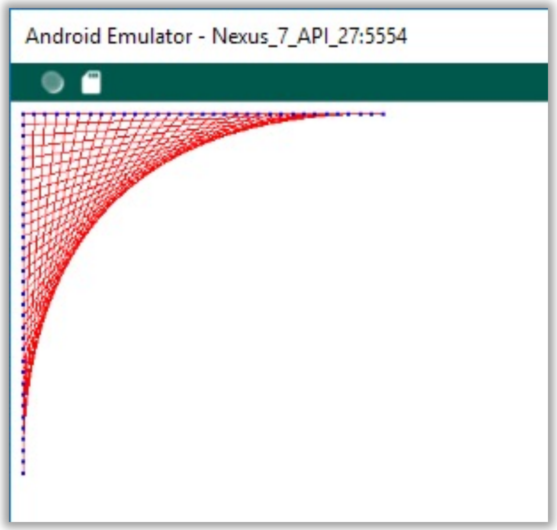


Category	Methods	Description
Creation	Constructors	There is a form of the constructor that are called when the view is created from code and a form that is called when the view is inflated from a layout file. The second form should parse and apply any attributes defined in the layout file.
	`onFinishInflate()`	Called after a view and all of its children has been inflated from XML.
Layout	`onMeasure(int, int)`	Called to determine the size requirements for this view and all of its children.
	`onLayout(boolean, int, int, int, int)`	Called when this view should assign a size and position to all of its children.
	`onSizeChanged(int, int, int, int)`	Called when the size of this view has changed.
Drawing	`onDraw(android.graphics.Canvas)`	Called when the view should render its content.
Event processing	`onKeyDown(int, KeyEvent)`	Called when a new hardware key event occurs.
	`onKeyUp(int, KeyEvent)`	Called when a hardware key up event occurs.
	`onTrackballEvent(MotionEvent)`	Called when a trackball motion event occurs.
	`onTouchEvent(MotionEvent)`	Called when a touch screen motion event occurs.
Focus	`onFocusChanged(boolean, int, android.graphics.Rect)`	Called when the view gains or loses focus.
	`onWindowFocusChanged(boolean)`	Called when the window containing the view gains or loses focus.
Attaching	`onAttachedToWindow()`	Called when the view is attached to a window.
	`onDetachedFromWindow()`	Called when the view is detached from its window.
	`onWindowVisibilityChanged(int)`	Called when the visibility of the window containing the view has changed.

https://developer.android.google.cn/reference/android/view/View?hl=en

The example

- In the next example, we will create a customized `view` called `MyView`.
  - Add a few lines and a few points to this view.
  - And put it into `MyViewActivity`.



```
class MyView extends View {
    private float[] mPts;
    private static final float DRAW_SIZE = 300;
    private static final int LINE_NUM = 32;

    public MyView(Context context) {
        super(context);
        buildPoints();
    }

    private void buildPoints() {
        final int ptCount = (LINE_NUM + 1) * 2;
        mPts = new float[ptCount * 2];
        float value = 0;
        final float delta = DRAW_SIZE / LINE_NUM;
        for (int i = 0; i <= LINE_NUM; i++) {
            mPts[i*4 + 0] = DRAW_SIZE - value;
            mPts[i*4 + 1] = 0;
            mPts[i*4 + 2] = 0;
            mPts[i*4 + 3] = value;
            value += delta;
        }
    }
```

MyView, First Half

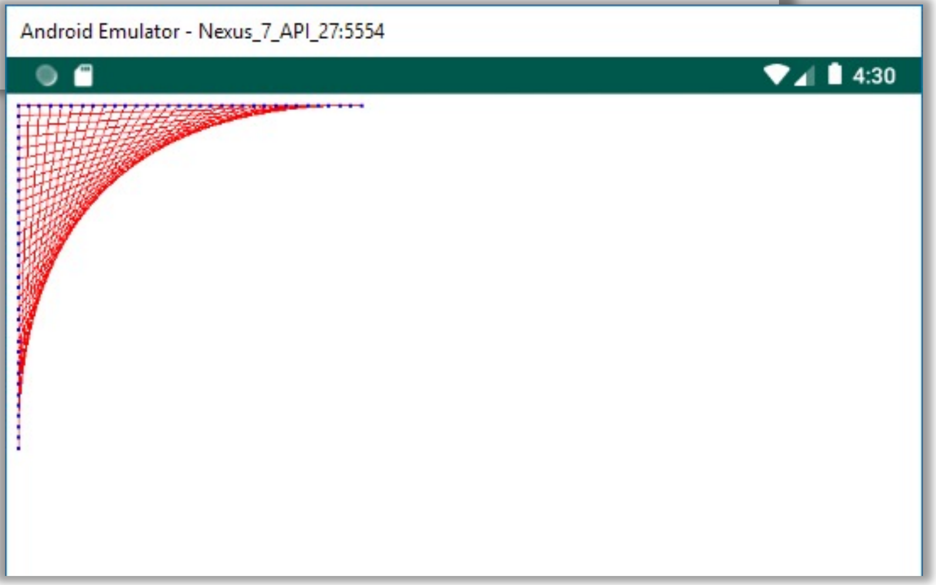
```
…
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.translate(10, 10);

    canvas.drawColor(Color.WHITE); // fill the background
    Paint p = new Paint();
    p.setColor(Color.RED);
    p.setStrokeWidth(0); // 0 means single pixel wide
    canvas.drawLines(mPts, p);

    p.setColor(Color.BLUE);
    p.setStrokeWidth(3);
    canvas.drawPoints(mPts, p);
}
```

(x, y) of this canvas will be (10 + x, 10 + y) of the View

MyView, Second Half

Code for MyViewActivity

```
public class MyViewActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
}
```



Summary of Steps

- Create your own `View` subclass.
- Override onDraw() method in your View.
- Use the canvas object passed from onDraw() to draw everything you want.
  - Use `Paint` to set the colur, style etc..

- You `View` subclass is ready, just add it inside an `Activity`.

Animated View

Update views using `view.invalidate()` and `view.postInvalidate()`

Animated View

- The drawing of a View uses the UI thread.

- In order to force a View to be redrawn:
  - You call `view.invalidate()` in the UI thread
  - Or call `view.postInvalidate()` in other thread.

- We can achieve animation using the above two methods (mostly `postInvalidate()`).

Animated View: Example

- In the next example, an animated View will be created.
- Read the code, can you explain the process of it?

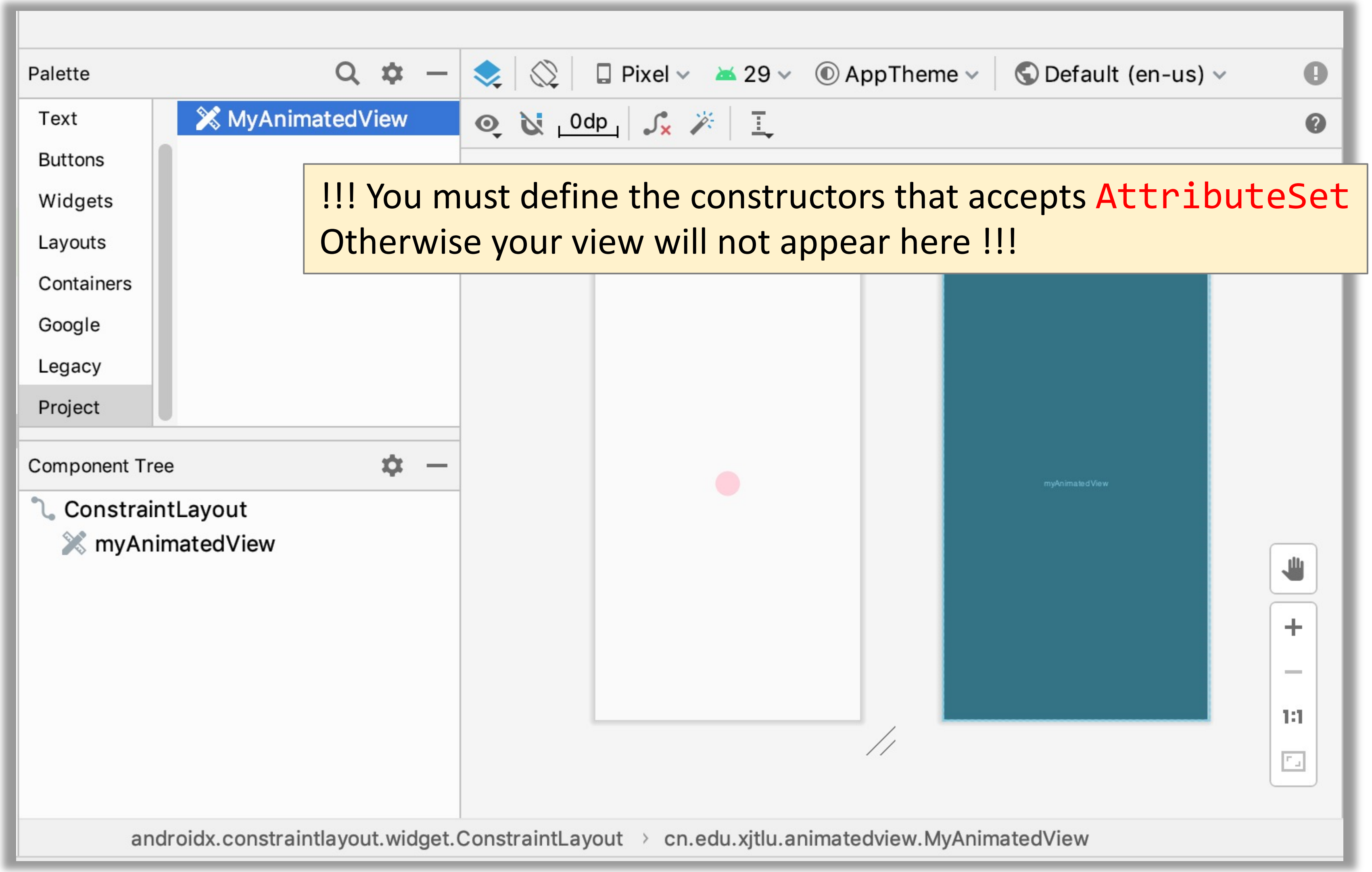
```java
public class MyAnimatedView extends View {
    public MyAnimatedView(Context context) {
        super(context);
    }

     /*
     * You must add one of the following two constructors if
     * you want this view to be available in the
     * UI designer of the Android Studio
     */
    public MyAnimatedView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public MyAnimatedView(Context context,
                          AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }
...
```

AnimatedView: Constructors

```
...
 float circleR = 50.0f;

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        int circleX = MyAnimatedView.this.getWidth() / 2;
        int circleY = MyAnimatedView.this.getHeight() / 2;


        Paint p = new Paint();
        p.setColor(0xaaFFBBCC);
        canvas.drawCircle(circleX, circleY, circleR, p);
    }
} // end of MyAnimatedView
```

Alpha, red, green and blue values

AnimatedView: onDraw()



!!! You must define the constructors that accepts AttributeSet
Otherwise your view will not appear here !!!

androidx.constraintlayout.widget.ConstraintLayout › cn.edu.xjtlu.animatedview.MyAnimatedView

Adding AnimatedView to Layout

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MyAnimatedView v = findViewById(R.id.myAnimatedView);
        AnimationThread t = new AnimationThread(v);
        t.start();
    }
}
```

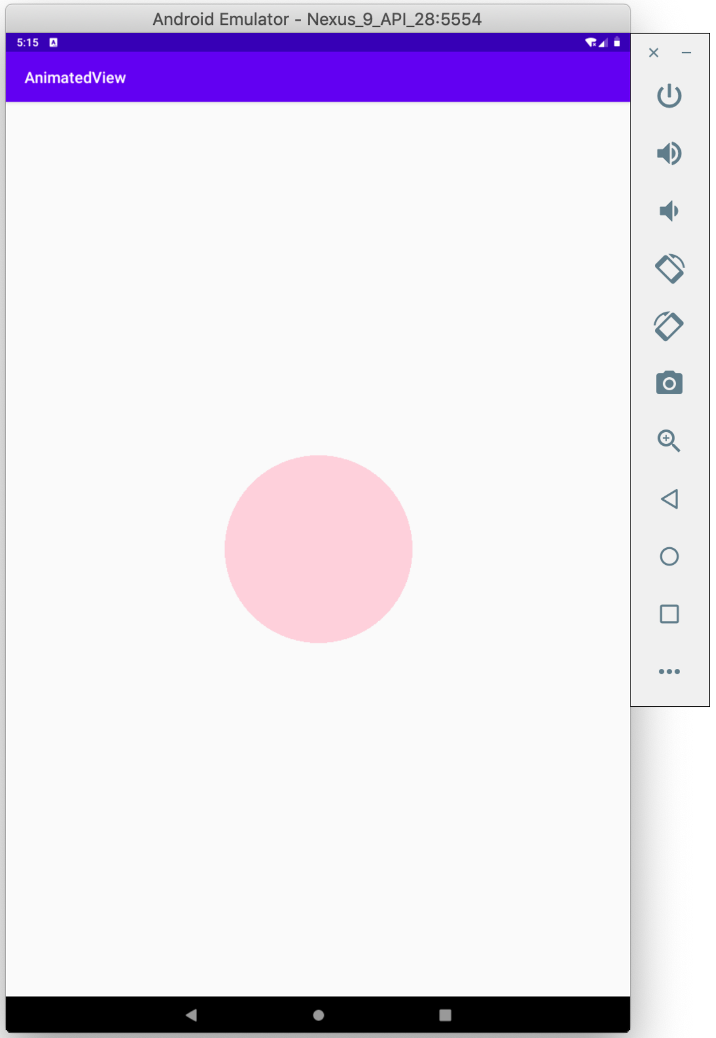
Finish the Activity

```java
class AnimationThread extends Thread {
    MyAnimatedView animatedView;
    public AnimationThread(MyAnimatedView view) {
        this.animatedView = view;
    }
    @Override
    public void run() {
        while (true) {
            int maxR = animatedView.getWidth() / 4;
            int minR = maxR / 2;
            animatedView.circleR = animatedView.circleR < minR ?
                                     maxR : animatedView.circleR - 20.0f;
            animatedView.postInvalidate();
            try {
                Thread.sleep(250);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

The Animation Thread

Animated View: Example

- It shows a circle that shrinks over time.
- The circle's Size is controlled by a separate thread.

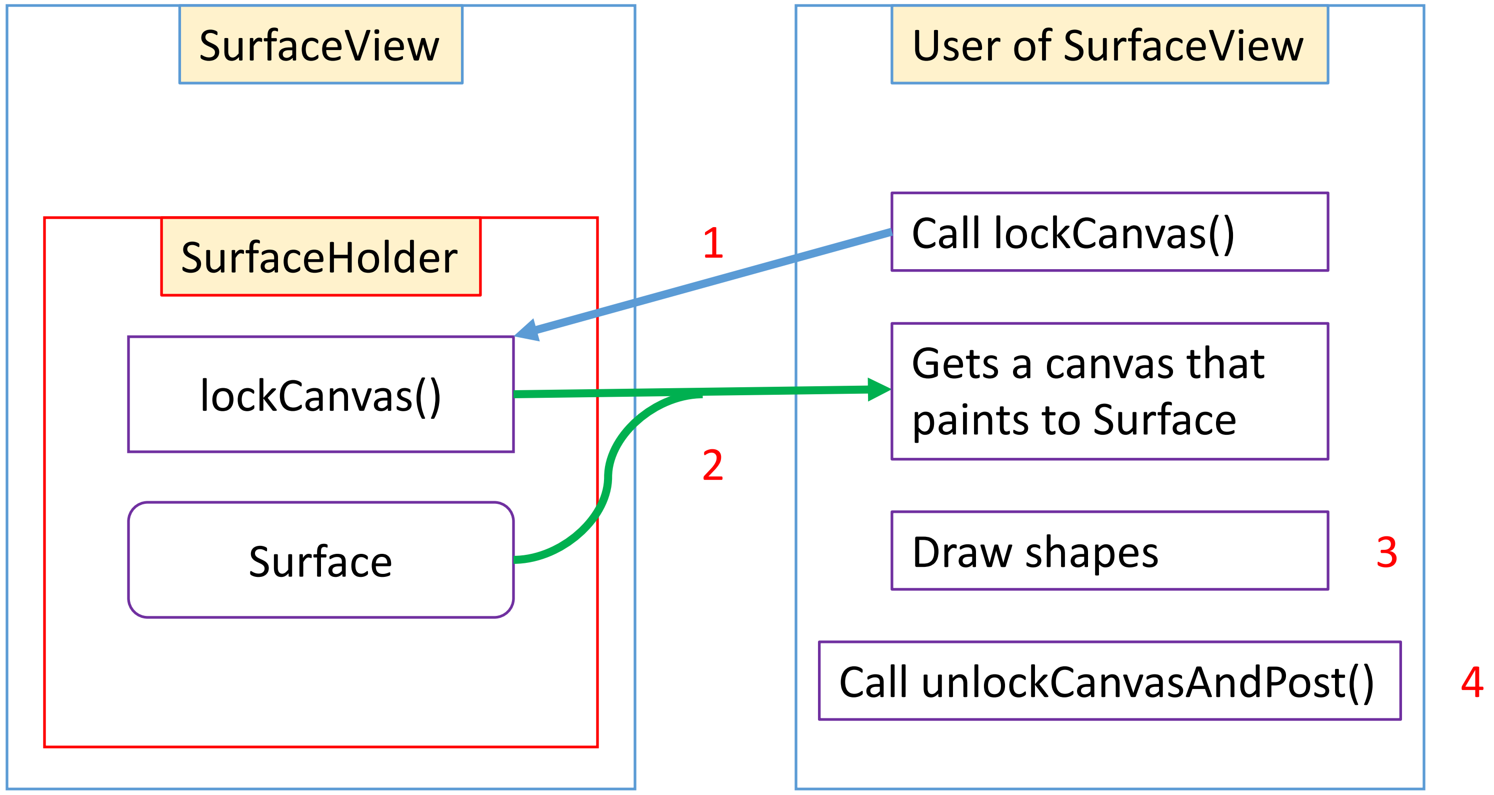


SurfaceView

SurfaceView and Related Classes

- `Surface`: “Handle on to a raw buffer that is being managed by the screen compositor”. This is where shapes and lines are drawn.
- `SurfaceHolder`: Contains a `Surface` object inside and can provide you a `Canvas` for drawing on the `Surface`.
- `SurfaceView`: Takes care of placing the `surface` at the correct location on the screen.
  - It has a `Surfaceholder` object that can be obtained using `getHolder()`.

Drawing to SurfaceView



Code Snippet

Get canvas and apply lock

```
Canvas canvas;
if ((canvas = surfaceHolder.lockCanvas()) == null)
{
    // can happen when the app is paused.
    continue; // next loop
}
canvas.drawColor(Color.WHITE);
…
surfaceHolder.unlockCanvasAndPost(canvas);
```

Release the lock

A full example will be available later

About lockCanvas()

- If a `Canvas` is returned, the canvas will be locked until `unlockCanvasAndPost(Canvas)` is called.
    - The returned `Canvas` can be used to draw into the surface's bitmap.

- A `null` is returned:
    - If the surface has not been created.
    - Or if the surface cannot be edited. (E.g. when your app is paused)

SurfaceHolder.Callback

- A `null` is returned:
  - If the surface has not been created.
  - Or if the surface cannot be edited. (E.g. when your app is paused)
- We cannot prevent an app from being paused or being put into the background. You can use a thread to call this function at a certain frequency.
  - But if you call `lockCanvas()` too often when the surface is not ready, your calls will be throttled to a slow rate in order to avoid consuming CPU.

SurfaceHolder.Callback

- A `null` is returned:
  - If the surface has not been created.
  - Or if the surface cannot be edited. (E.g. when your app is paused)

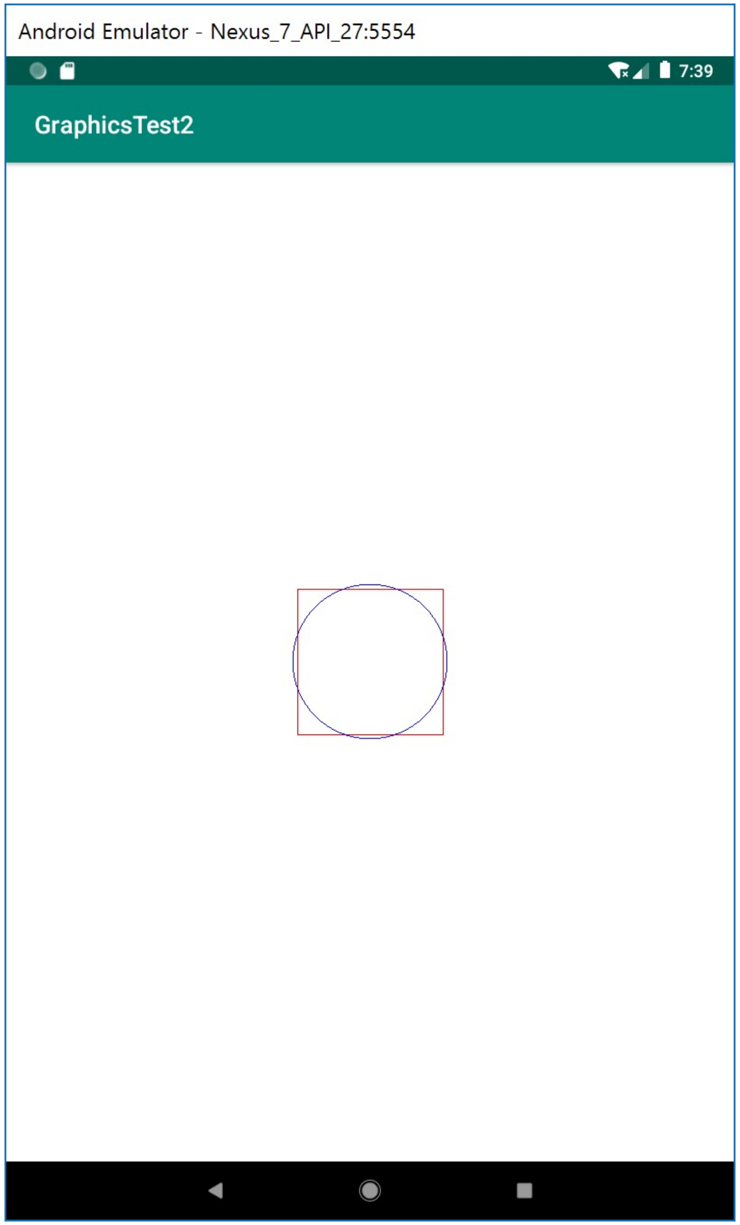
- For this case, we can use a function called `SurfaceHolder.addCallback()` to add an `SurfaceHolder.Callback` object.
  - The corresponding callback functions of this object will be called automatically by the Android OS.

Class SurfaceHolder.Callback

- The call back functions are:
  - `surfaceChanged()`
    This is called immediately after any structural changes (format or size) have been made to the surface.
  - **`surfaceCreated()`**
    This is called immediately after the surface is first created.
  - `surfaceDestroyed()`
    This is called immediately before a surface is being destroyed.

The Next Example

- In the next example, we will use `SurfaceView` to display a square and a circle.
- A thread will be running at the background, constantly updating the image.
- Read the code, and answer: What will the animation look like?



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <SurfaceView
        android:id="@+id/surfaceView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Adding a SurfaceView in Layout.XML

```
public class MainActivity extends AppCompatActivity {
    private SurfaceView mySurfaceView;
    private SurfaceHolder surfaceHolder;
    private DrawingThread t = new DrawingThread();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mySurfaceView = findViewById(R.id.surfaceView);
        surfaceHolder = mySurfaceView.getHolder();
        surfaceHolder.addCallback(new SurfaceHolder.Callback() {
            @Override
            public void surfaceCreated(SurfaceHolder surfaceHolder) {
                t.start();
            }
            @Override
            public void surfaceChanged
                (SurfaceHolder surfaceHolder, int i, int i1, int i2) {}

            @Override
            public void surfaceDestroyed(SurfaceHolder surfaceHolder) {
            }
        });
    }
…
```

Start the animation thread

Setting up

```java
class DrawingThread extends Thread {
    final int CIRCLE_TICK_REVERT = 15;
    @Override
    public void run() {
        for(int tick = 0; doDrawing; tick = (tick + 1) % CIRCLE_TICK_REVERT) {
            Canvas canvas;
            if ((canvas = surfaceHolder.lockCanvas()) == null) {
                continue;
            }
            canvas.drawColor(Color.WHITE);
            final int midX = canvas.getWidth()/2;
            final int midY = canvas.getHeight()/2;
            final int RADIUS = midX/5;
            Paint p = new Paint();
            p.setStrokeWidth(0);
            p.setStyle(Paint.Style.STROKE);
            p.setColor(Color.RED);
            canvas.drawRect(new Rect(midX - RADIUS, midY - RADIUS,
                                     midX + RADIUS, midY + RADIUS), p);
            p.setColor(Color.BLUE);
            int circleR = RADIUS * (10 + CIRCLE_TICK_REVERT - tick) / 15;
            canvas.drawCircle(midX, midY, circleR, p);
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {}
            surfaceHolder.unlockCanvasAndPost(canvas);
        }
    }
}
```

This thread class is defined inside our activity class

Some Important Properties

- The content of the `Surface` is never preserved between `unlockCanvas()` and `lockCanvas()`, for this reason, every pixel within the Surface area must be written.

- Drawing to `View` uses the UI thread. But drawing to `SurfaceView` can be done in a separate thread.
  - The performance of the UI thread might be affected if you achieve animation using view

View & SurfaceView

More discussions on the difference between drawing on `View` and drawing on `SurfaceView`

https://stackoverflow.com/questions/1243433/difference-between-surfaceview-and-view

Lab

- Please go through the examples in this lecture first.
- Then, create a program that shows a moving ball.
  - The ball moves at a constant speed and towards a random direction.
  - The ball bounces when it reaches the edge of the View.
  - Add a button, when this button is clicked, a new bouncing ball will be added with another random moving direction.
  - Up to 3 balls can be added.
  - You can use either `View` or `SurfaceView`.