# Mobile 3D Graphics

## Introduction to Android Views

# Custom Views

- The Android framework provides several **default views.**
- The base **class a view** is the **View**.
- **Views** are responsible for **measuring**, **layouting** and **drawing** themselves and their child elements.
- **Views** are also responsible for **saving their UI state** and **handling touch events.**
- Developers can also create **Custom Views** and use them in their application.

# Create Custom Views

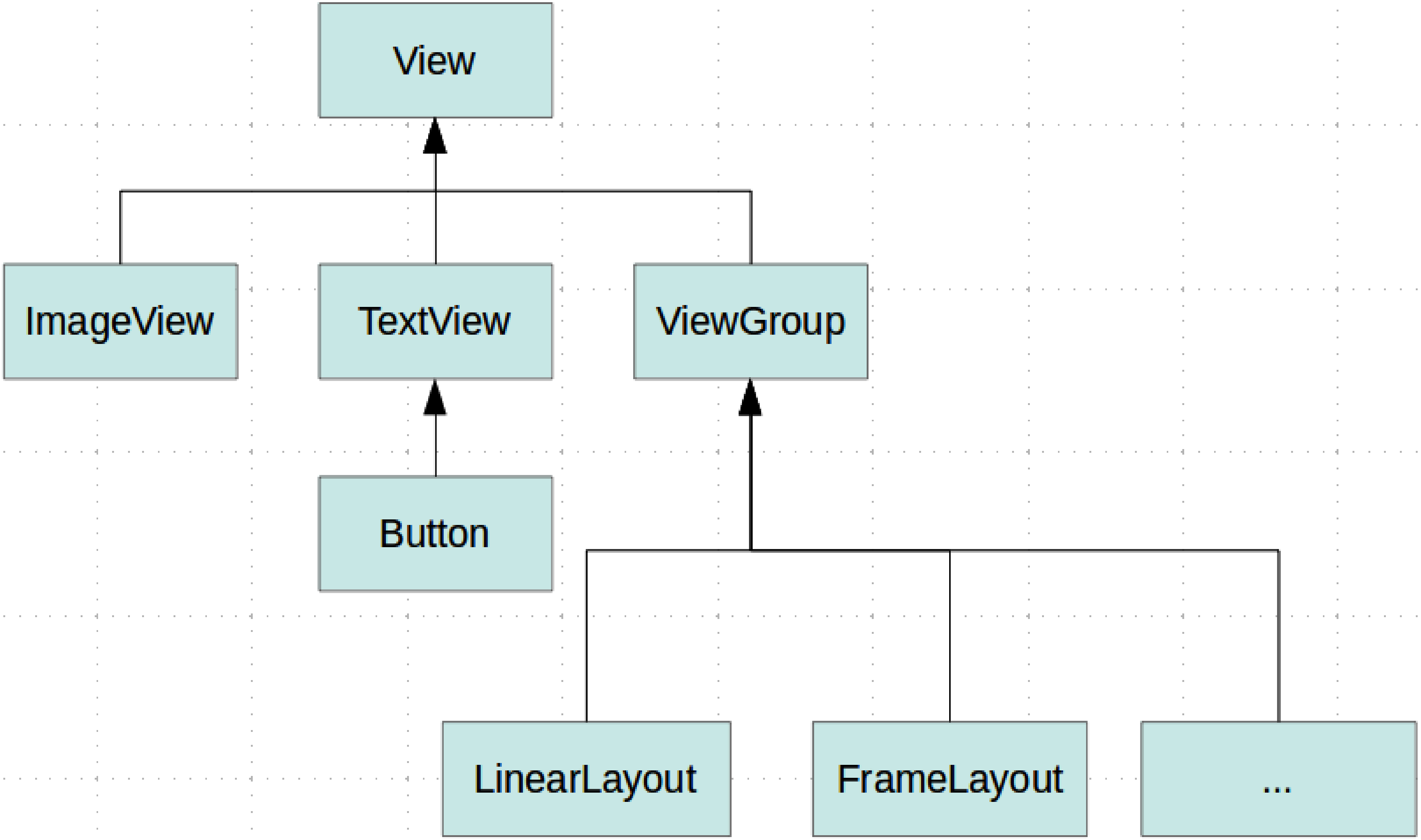
**It is possible to create Custom Views by:**

- **Compound views** - combining views with a default wiring.
- **Custom views** - creating your own views
  - ➢ by **extending** an existing **view**, **e.g. Button**
  - ➢ by **extending** the **View class**

# Compound Views

- **Compound views** (also known as ***Compound Components*** ) are pre-configured **ViewGroups** based on existing **views** with some predefined **view interaction**.
- **Compound views** also allow you to add **custom API** to **update** and **query** the state of the compound view.

# Default View Hierarchy Of Android



# How Android draws the view hierarchy

- Once an **activity** receives the focus, it must provide the **root node** of its **layout hierarchy** to the Android system. Afterwards the **Android system starts the drawing procedure.**
- **Drawing** begins with the root node of the **layout**.

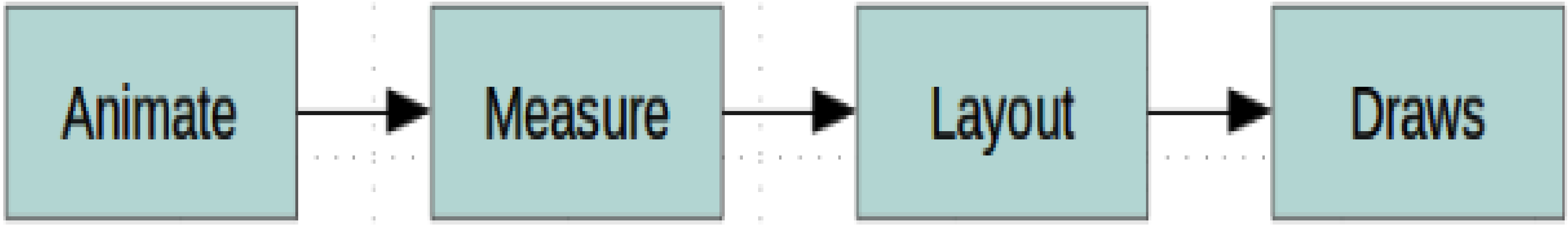
**Drawing the layout is a two pass process:**

- **measuring pass** - implemented in the **`measure(int, int)`** method. Every view stores its measurements.
- **layout pass** - implemented in the **layout(int, int, int, int)** method. During this phase each **layout manager** is responsible for positioning all of its children. It uses the sizes computed in the **measure pass**.

# Life cycle of a Android view

## Traversal life cycle events

- **onMeasure()** method determines the **size** for the **view** and its children.
- **onLayout()** positions the **views** based on the result of the **onMeasure()** method call.



# Creating custom views

- By extending the **View class** or one of its **subclasses** you can create your **custom view**.
- For drawing view use the **onDraw()** method. In this method you receive a **Canvas object** which allows you to perform drawing operations on it, e.g. **draw** **lines, circle, text** or **bitmaps**.

# Using new views in layout files

- **Custom** and **compound** views can be used in layout files.
- For this you need to use **the full qualified name in the layout file**, **e.g.** using the **package** and **class** name.

```
<?xml version="1.0" encoding="utf-8"?>
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

     <de.vogella.android.ownview.MyDrawView
         android:id="@+id/myDrawView1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />

 </LinearLayout>
```

# Define additional attributes for your custom Views

- To define additional attributes create an *attrs.xml* file in your *res/values* folder.
- The following shows an **example** of attributes defined for a new view called **ColorOptionsView**.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <declare-styleable name="ColorOptionsView">

        <attr name="titleText" format="string“   localization="suggested" />

        <attr name="valueColor" format="color" />

     </declare-styleable>

</resources>
```

# Create A Custom View

There are **TWO WAYS** of making custom views in Android:

1. Extending the **View class-** Building the view from scratch
2. Extending **already existing views ( TextViews, LinearLayouts Etc )**

**In this Exercise**, we will be focusing on the first way of making custom views.

1. How to make ***basic shapes*** using Custom Views
2. How to add ***custom attributes*** to your Custom Views
3. How to make ***shape manipulations*** using Custom Views ***(increase/decrease shape size, change shape color using functions)***

# Exercise1:
# Make Basic Shapes

1. Create a new *Android Studio project* and select *Empty Activity* template. At this point you should only have one class named *"MainActivity"* inside your project.
2. Create a new class, name it *"MyCustomView"***,** and extend in by ***View class.***
3. **At this point**, android Studio will prompt you to an **error** to create ***constructor(s) matching super***. **On clicking the prompt**, you should select all the options for the constructor.

# Make Basic Shapes cont

4. create a new function ***void init(@Nullable AttributeSet set)*** with **blank body** and make all the constructors access this function by calling ***init(attrs)*** on all constructors (except you have to pass **null** in the first constructor)

5. **Override** the ***onDraw(Canvas canvas)*** in this class. **In this function you have to:**

- Create a new ***Paint*** object and assign a **color** to it,
- Create a ***Rect*** object and assign ***left, right, top, bottom*** coordinates to it
- then call ***canvas.drawRect( your rect object, your paint object).***

6. **Last step:** Add your **custom view** to the ***activity_main.xml.***

```java
package com.example.dell.g_custom_view;

    import android.content.Context;
    import android.graphics.Canvas;
    import android.graphics.Color;
    import android.graphics.Paint;
    import android.graphics.Rect;
    import android.os.Build;
    import android.support.annotation.Nullable;
    import android.support.annotation.RequiresApi;
    import android.util.AttributeSet;
    import android.view.View;

    public class MyCustomView extends View {
        public MyCustomView(Context context) {
            super(context);
            init(null);
        }

        public MyCustomView(Context context, @Nullable AttributeSet attrs) {
            super(context, attrs);
            init(attrs);
        }
```

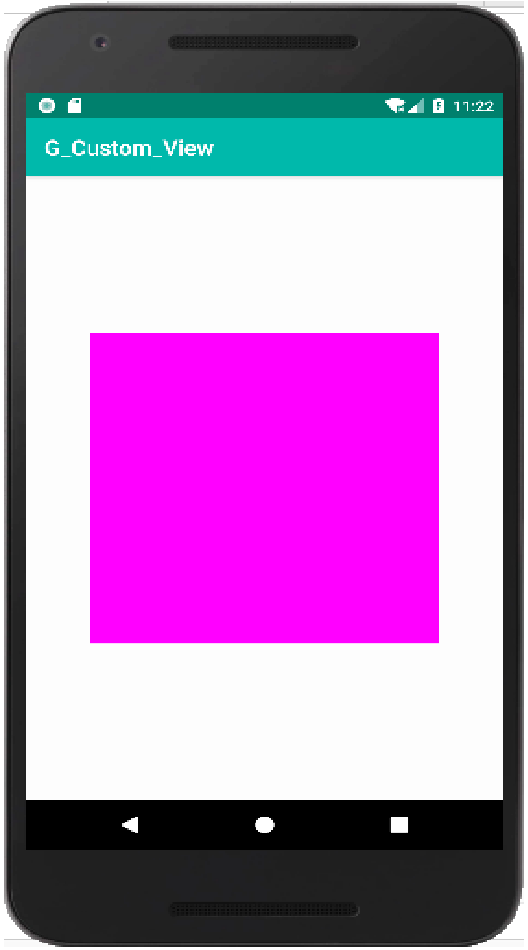
```
public MyCustomView(Context context, @Nullable AttributeSet attrs, int
  defStyleAttr) {
        super(context, attrs, defStyleAttr);
        init(attrs);
    }

    @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
    public MyCustomView(Context context, @Nullable AttributeSet attrs, int
  defStyleAttr, int defStyleRes) {
        super(context, attrs, defStyleAttr, defStyleRes);
        init(attrs);
    }

    private void init(@Nullable AttributeSet set) {
    }
```

```
@Override
    protected void onDraw(Canvas canvas) {
      super.onDraw(canvas);
      Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
      paint.setColor(Color.MAGENTA);
      Rect rect = new Rect();
      rect.left = 0;
      rect.right = getWidth();
      rect.top = 0;
      rect.bottom = getHeight();

      canvas.drawRect(rect, paint);
    }
 }
```



# Add your custom view
# *activity_main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:gravity="center"
    tools:context="com.example.dell.g_custom_view.MainActivity">

    <com.example.dell.g_custom_view.MyCustomView
        android:layout_width="300sp"
        android:layout_height="300sp" />

    </LinearLayout>
```

# Exercise2:
# Add Custom Attributes

1. Make ***mRect and mPaint*** objects of **Rect** and **Paint** class respectively as ***global*** to the class. make their ***instances in the init() method*** that was made. Then replace ***rect with mRect, and paint with mPaint.***The warning should be removed by following this step.

```
● public class MyCustomView extends View{
        Paint mPaint;
        Rect mRect;
        int mSquareColor;
```

# Exercise2:
# Add Custom Attributes

2. Now, to begin adding ***custom attributes*** to your custom views, you have to first add a ***new file your "values" directory and name it "attrs.xml".*** Inside this xml file, ***inside <resources> </resources> tags, add a "declare-styleable" tag with attribute "name" as MyCustomView*** (your custom view class name).

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="MyCustomView">
        <attr name="square_color" format="color"/>
    </declare-styleable>
</resources>
```

3. **Inside these tags**, all your custom attributes will be inserted in the form of ***key ("name=") — value ("format=")*** pairs. In our case, we will add a custom attribute named ***square_color*** with format as ***color.***

# Add Custom Attributes cont

4. **Next**, we need to check in our ***init() method*** whether the **AttributeSet *set*** being passed as a parameter is **null or not**. If it is not null, then we obtain a ***TypedArray typedArray (say)*** by calling ***obtainStyledAttributes(set, R.styleable.MyCustomView)*** using **getContext()**;

5. **Next**, we declare an **int variable *mSquareColor*** and initialise with the values input through the **TypedArray ta**, also providing the default **color**. Also remember to call ***ta.recycle()*** once you are done accessing it.

```
private void init(AttributeSet set){
    mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    mRect = new Rect();


    if(set == null){
        return;
    }
  TypedArray ta = getContext().obtainStyledAttributes(set, R.styleable.MyCustomView);
   mSquareColor = ta.getColor(R.styleable.MyCustomView_square_color, Color.GREEN);
   mPaint.setColor(mSquareColor);
    ta.recycle();
 }
```

# Add Custom Attributes cont

6. **Now** all you need to do is add your custom attribute ***square_color*** to your ***activity_main.xml*** , you will see that the custom view color changes to whatever color you add inside the attribute parameter.

```
<com.example.dell.g_custom_view2_attributes.MyCustomView
        android:layout_width="300sp"
        android:layout_height="300sp"
        app:square_color="@color/colorPrimary"/>
```

- **More examples on custom attributes are for size of your view, radius in case of circle, text input, etc.**

# Exercise:
# Create A Compound View

# References