# A Drawing Android App

### Aims:

To introduce developing an Android App for interactive and dynamic graphics

### Objectives:

* Extending a View class
* Using the Canvas class
* Using the Paint Class
* Drawing into a Bitmap
* Realising an OnTouchListener interface

## 1. Introduction

Here we’ll use Android Studio to develop an interactive graphics app using a custom view.

**2. Starting a new project**

Let’s get started…

From windows start Android Studio and start a new Android Studio project.

Choose an **Empty Activity**, name the project ‘**MyDrawingApp**’ and ensure the language is ‘**Java**’ before you click [Finish]. Wait for Gradle sync and indexing to finish (see bottom right of Android Studio).

From the Project explorer right click on …

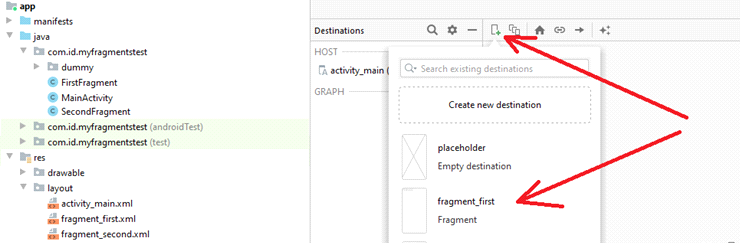
**app/java/com.*your-id*. mydrawingapp**

to create a **New / Fragment / Fragment (Blank)**. In the resulting dialog select the ‘Fragment Name:’ to be ‘**FirstFragment**’

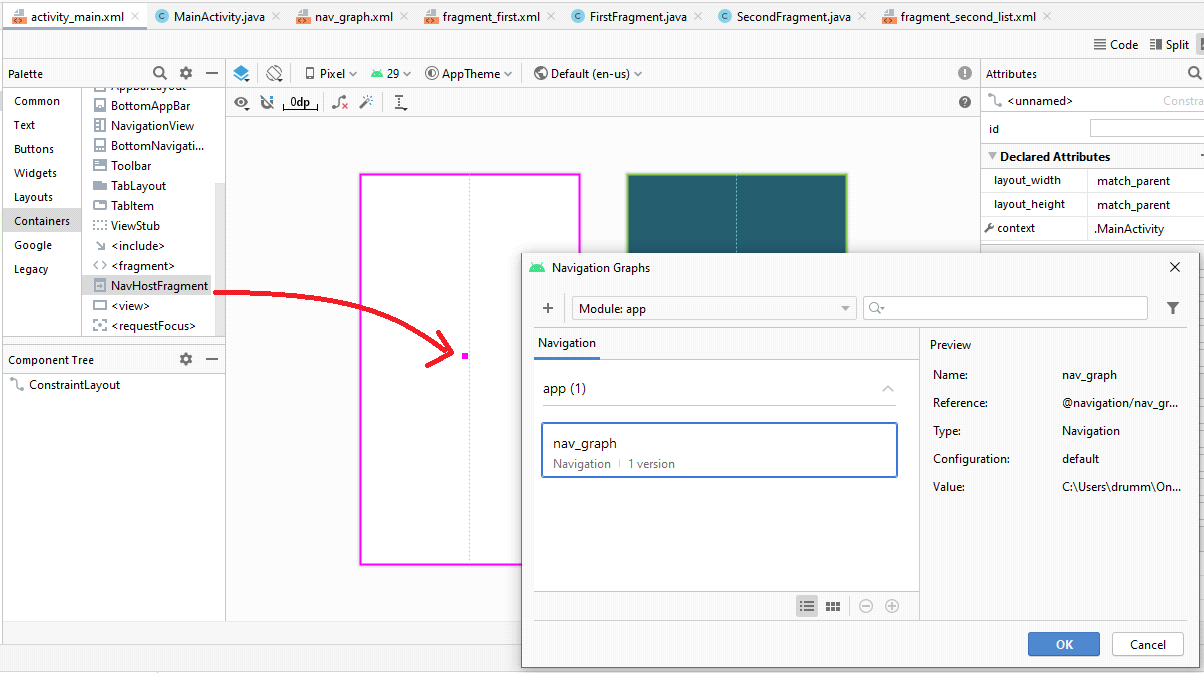
Right click on ‘**res**’ directory to create a new Android Resource File, from the resulting dialog set ‘File Name’: to be ‘**nav\_graph**’ and select ‘Resource type :’ to be ‘navigation’.

You maybe be prompted with a message to add additional libraries, click [OK].

In the Design View you will see an empty navigation graph. Hence, choose to add your **fragment\_first** to the graph



Go to **activity\_main.xml** file, delete the TextView, and hence drag and drop from Containers a **NavHostFragment** onto your design view, hence when prompted, choose the **nav\_graph** you created earlier.



From the nav\_host\_fragment’s attribute inspector, change…

* the id to **nav\_host\_fragment**.
* **layout\_width** and **layout\_height** to **match\_parent**.

## 3. Sub classing the View class

There are a number of ways to draw things to an android app. Typically you will use the drawing methods associated with the Canvas class and use the Paint class to define your Brush. Here we’ll extend the View class and hence toggle between this and our initial layout using our options menu.

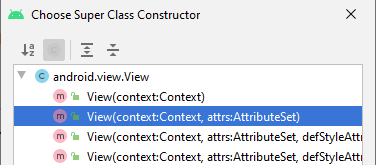
From the Android Explorer right click on ‘app/java/com.*yourpackage*.drawingapp’ to create a New->Java Class called MyView.

In the resulting MyView.java file you’ll have a new class defined…

**package** com.example.ian.drawingapp;

**public class** MyView {  
}

Subclass View using the ‘extends’ keyword. Let Android Studio ‘Import View (view.android)’ and ‘Add constructor MyView(context:Context, attrs:AttributeSet)’ *… using [Alt Enter]*.



Your class definition should now look like…

**public class** MyView **extends** View {  
**public** MyView(Context context, @Nullable AttributeSet attrs) {  
 **super**(context, attrs);  
 }  
}

In MyView define as fields

Paint paint;

**float** x=100,y=100;

And in the MyView constructor method instantiate a Paint object and set some values

paint = **new** Paint();

paint.setColor(Color.***YELLOW***);

paint.setTextSize(100);

To do drawing we can override the View’s onDraw(Canvas) method.

From Android Studio’s main menu choose ‘**Code->Override Methods …**’ add to the MyView class…

@Override

**protected** **void** onDraw(Canvas canvas) {

**super**.onDraw(canvas);

}

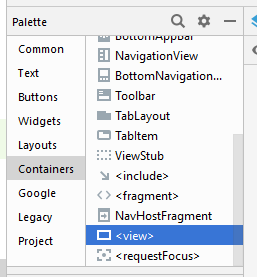
And add to this a call to a member method of canvas to draw some text

canvas.drawText("Hello", x, y, paint);

Go to **first\_fragment.xml**

Right click on the frame layout in the component tree to convert is to a constraint layout.

Drag a text view onto the design area and set its text to “Draw something in canvas”. You might also want to change the text size.



Drag a <view> onto the design area for this fragment. A dialog box will offer you a choice of classes to associate with this, choose MyView.

Constrain these views horizontally and vertically.

#### 

#### EXERCISE

From MyDrawable’s draw() method add more graphic primitives including lines, rectangles and text of different sizes and colours. Hence with code paint a picture.

#### 4. Interaction

Let’s try adding some interaction by touching the screen (though with the mouse in emulator).

We’ll use x and y to control the position of something to be drawn at the touch position.

In the MyView class we’ll define a callback to handle touch events.

Add to the MyView class **as a field** a realisation of the ‘OnTouchListener’ interface.

OnTouchListener t1Listener = **new** OnTouchListener() { };

Remember the brackets indicate that we are not instantiating an object of type OnTouchListener, but instead instantiating an anonymous class that realises the OnTouchListener interface.

Let Android Studio add the import statement and add the unimplemented method to give…

OnTouchListener t1Listener = **new** OnTouchListener() {

@Override

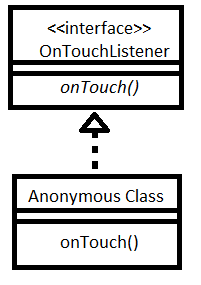
**public** **boolean** onTouch(View v, MotionEvent event) {

// **TODO** Auto-generated method stub

**return** **false**;

} };

Here’s the UML class diagram for the above code.



Replace the ‘return false’ in the outTouch() method with code to set myView’s x and y to the touch event’s x and y. The invalidate() will schedule a call to MyView’s onDraw() method.

x=event.getX();

y=event.getY();

v.invalidate();

**return** **true**;

In MyView’s constructor method associate ‘t1Listener’ with the view.

setOnTouchListener(t1Listener);

Next declare as fields in MyView…

Bitmap **myBitmap** = **null**;  
Canvas **myBitmapCanvas** = **null**;  
**int width**=800, **height**=800;

From Android Studio’s main menu choose ‘**Code->Override Methods …**’ add to the MyView class…

@Override  
**protected void** onMeasure(**int** widthMeasureSpec, **int** heightMeasureSpec) {  
 **super**.onMeasure(widthMeasureSpec, heightMeasureSpec);  
  
}

Add to this method code for instantiating a bitmap and an associated canvas of the size of the **<view>** you added to your UI…

**width**=getMeasuredWidth();  
**height**=getMeasuredHeight();  
**myBitmap** = Bitmap.*createBitmap*(**width**,**height**,Bitmap.Config.***RGB\_565***);  
**myBitmapCanvas** = **new** Canvas(**myBitmap**);

Hence in the **onDraw()** method, draw to the bitmap with for example

myBitmapCanvas.drawCircle(x, y,10,paint);

And finally draw the bitmap to the drawable with

canvas.drawBitmap(myBitmap,0,0,**null**);

Build and Run.



**EXERCISE**

For a smoother looking drawing experience, explore the nature of touch events. For example, instead of drawing circles you could draw lines between points using

**myBitmapCanvas**.drawLine(**x**, **y**, **prevX**, **prevY**, **paint**);  
**prevX**=**x**;  
**prevY**=**y**;

Every time you begin dragging you finger around the screen set the previous x and y to the current x and y by testing for ACTION\_DOWN events.

**if** (event.getAction() == event.***ACTION\_DOWN***) {  
 **prevX** = **x**;  
 **prevY** = **y**;  
}

**EXERCISE**

If you have time, add to the first fragment’s design some UI controls such as buttons that let the user change the drawing colour. See if you can figure out how to do this yourself.

Clue, in the FirstFragment.java file’s FirstFragment class

* override the **onViewCreated()** method …
* get a reference to myView via view.findViewById(…);

MyView **myView**;  
  
@Override  
**public void** onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 **super**.onViewCreated(view, savedInstanceState);

myView = view.findViewById(R.id.***you\_view\_id***);  
  
 myView.*somePublicMethodYouDefined()*;  
}

* get references to your buttons, etc via view.findViewById(…);
* set OnClickListeners for the buttons, or other UI controls
* and call setter methods which you can write in the MyView class to change paint colours.

Try to create a well featured drawing app.

**COMPLETE CODE EXAMPLE**

**MyView.java**

**public class** MyView **extends** View {  
 Paint **paint**;  
 **float x**=100,**y**=100;  
 Bitmap **myBitmap** = **null**;  
 Canvas **myBitmapCanvas** = **null**;  
 **int width**=800, **height**=800;  
  
 OnTouchListener **t1Listener** = **new** OnTouchListener() {  
 @Override  
 **public boolean** onTouch(View v, MotionEvent event) {  
 **x**=event.getX();  
 **y**=event.getY();  
 v.invalidate();  
 **return true**;  
  
 }  
 };  
  
 **public** MyView(Context context, AttributeSet attrs) {  
 **super**(context, attrs);  
 **paint** = **new** Paint();  
 **paint**.setColor(Color.***YELLOW***);  
 **paint**.setTextSize(100);  
 setOnTouchListener(**t1Listener**);  
 }  
  
 @Override  
 **protected void** onDraw(Canvas canvas) {  
 **super**.onDraw(canvas);  
 **myBitmapCanvas**.drawCircle(**x**, **y**,10,**paint**);  
 canvas.drawBitmap(**myBitmap**,0,0,**null**);  
 }  
  
 @Override  
 **protected void** onMeasure(**int** widthMeasureSpec, **int** heightMeasureSpec) {  
 **super**.onMeasure(widthMeasureSpec, heightMeasureSpec);  
 **width**=getMeasuredWidth();  
 **height**=getMeasuredHeight();  
 **myBitmap** = Bitmap.*createBitmap*(**width**,**height**,Bitmap.Config.***RGB\_565***);  
 **myBitmapCanvas** = **new** Canvas(**myBitmap**);  
 }  
}

**fragment\_first.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:id="@+id/firstLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_centerHorizontal="false"  
 android:layout\_centerVertical="false"  
 tools:context=".FirstFragment"**>  
  
 <**TextView  
 android:id="@+id/textView2"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Draw Something In Canvas"  
 android:textSize="30sp"  
 app:layout\_constraintBottom\_toTopOf="@+id/view2"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"** />  
  
 <**view  
 android:id="@+id/view2"  
 class="com.id.mycustomview.MyView"  
 android:layout\_width="320dp"  
 android:layout\_height="554dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.5"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/textView2"** />  
</**androidx.constraintlayout.widget.ConstraintLayout**>