Lab 6: Custom Views

Download the ball.png image file from Moodle. You should copy ball.png, select the *res/drawable* folder when you have created your project and paste it there. You can now access this image for use in your app.

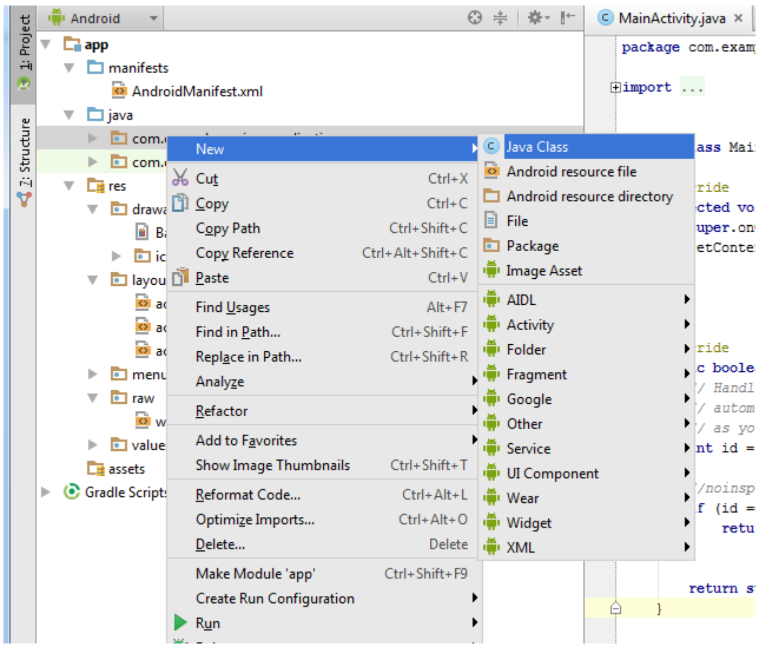
Displaying a simple image - ImageView

An ImageView is a simple way to include an image in an app whilst looking at a layout in Design view simply drag an imageView from the widgets palette onto the device screen. Double click the object and, to set the ball image as src, navigate to the images – you will need to scroll down to the drawable folder to select the ball. This will be shown in the render.

This way of displaying images is of limited use in games where it is usually necessary to refresh the screen, either regularly (turn based games) or frequently (more action games). This lab will introduce you to how to use Views to display a game interface for simpler types of games and next week's lab will look at the approach for when the screen needs to be refreshed rapidly.

Creating a Custom View

For turn based games that do not require a significant amount of processing or frame-rate speed (perhaps for a chess or card game), you should create a custom View component and drawing on its Canvas using onDraw(); The Android framework will provide you with a pre-defined Canvas to which you will place your drawing calls.



Create a new project. From now on we will call the main activity *GameActivity*.

What we now need to do is create the "view" that will display the game. We will use this view to set the contentView instead of a layout file as we did previously. You do not need to create a layout file for the activity.

Select the package folder in the java folder in your project. Right click and select create a new java class. Call this *GameView* and enter the code below underneath the package.

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.View;

public class GameView extends View {
        // declare variables needed for View
        private Paint redPaint;

        public GameView(Context context) {
                super(context);
                redPaint = new Paint();
                redPaint.setColor(Color.RED);
        }

        @Override
        protected void onDraw(Canvas canvas) {
                canvas.drawCircle(50, 50, 10, redPaint);
        }
}
```

Our GameView is extending View, and remember that Views can be associated with screens. There is an onDraw() method that does the drawing (drawing a circle in this case) and receives the canvas to draw on.

The Android framework calls the onDraw() method automatically when creating the GameView. Have a look at what we are trying to draw.
Now in the GameActivity you will need to make an instance of GameView, so declare a GameView after the class definition:

```
        GameView GV;
```

In onCreate() put:

```
        GV = new GameView(this);// creates the custom view – invoking the GameView constructor
        setContentView(GV);     //sets the content view to the game view and that is then displayed.
```

Delete the setContentView that refers to an XML layout file if you have to. Check that this displays a red circle. Where is its centre? What determines how big the circle is? Look up drawCircle() http://developer.android.com/reference/android/graphics/Canvas.html
drawCircle(float, float, float, android.graphics.Paint).

With the ability to draw on the canvas with a series of other draw methods (drawBitmap(), drawRect(), drawText() etc.) we are equipped to display game components.

Displaying a bitmap image is important in many games. So that there is a bitmap to use paste ball.png into the drawable folder of this project.

Firstly the bitmap needs to be declared in the GameView (and library imported)

```
        private Bitmap ball1;
```

This needs to be attached to the image in the drawable folder in the GameView's constructor (after super(context): `ball1 = BitmapFactory.decodeResource(getResources(), R.drawable.ball);`
and drawn in onDraw(): `canvas.drawBitmap(ball1, 10, 10, null);`

Test that this works and seek help if there are any problems. Notice that the file extension for the graphic is not used. Now to make a game we will need some kind of interaction – let us go for touch events on the screen. Before we do this it is worth setting up a drawText() to put some output on the screen to the redPaint attribute settings in the GameView constructor to increase the font size: `redPaint.setTextSize(48);`

Declare a string: `private String output = "output will appear here";`
In the onDraw() method: `canvas.drawText(output, 10, 100, redPaint);`

Now to set up some interaction. Include the following method in the GameView. This method will test whether a MotionEvent has occurred and show what type it is. The strange call to invalidate() will force the system to call the onDraw() method again and update screen.

```
public boolean onTouchEvent(MotionEvent event) {
        int eventaction = event.getAction();

        switch (eventaction ) {
                case MotionEvent.ACTION_DOWN:
                        output = "down";
                break;
                case MotionEvent.ACTION_MOVE:
                        output = "move";
                break;
                case MotionEvent.ACTION_UP:
                        output = "up";
                break;
        }
        invalidate();
        return true;
}
```

The two int variables at class level with represent coordinates for the ball:

```
        private int ballX, ballY;
```

Draw the ball at a position given by these coordinates:

```
        canvas.drawBitmap(ball1, ballX, ballY, null);
```

Before the switch statement in the onTouchListener():

```
        int X = (int)event.getX();
        nt Y = (int)event.getY();
```

Add in the ACTION_DOWN case statement.

```
        ballX = X;
        ballY = Y;
```

Check to see that the ball moves to where the screen is pressed. Investigate what difference it makes moving these last two lines into the other switch statements. Can you get the ball to be dragged?

You might have noticed that the ball is drawn from the top left corner of its graphic. To draw it with the X, Y coordinates at its centre replace the bitmap drawing line by:

```
canvas.drawBitmap(ball1, ballX - ball1.getWidth()/2, ballY - ball1.getHeight()/2, null);
```

Another thing we might want to do is get the ball to move when it is touched – effectively using a hit test. In the onTouchEvent() method we can test to see if the ball has been hit:

```
If (X > ballX && X < ballX + ball1.getWidth() && Y > ballY && Y < ballY + ball1.getHeight())
output = "hit";
```

For this to work properly you will need to revert to

```
canvas.drawBitmap(ball1, ballX, ballY, null);
```

in onDraw(). Can you see why?

If we want to move the ball the action to take might include setting:

```
ballX = (int)(Math.random()*720);
ballY = (int)(Math.random()*1280);
```

Exercise: For the second assessment you will have to create a game using a surfaceView. You will need to think of enhancements. Some ideas:
Graphics acting as buttons using hit tests – these buttons might display rules, start or stop a game or display a score. Or the score might update as the game is played. You might want to add sound or a win or lose screen. Or a timer. Or some other aspects of game play such as drag and drop or collisions between objects.