Week 10: Lab Tasks

A. Android Touch Event

A.0 Android Touch Events - Overview

Recommended Reading: Week 10 > Lecture Material

In this example, we will cover two ways to capture touch events. A view can intercept touch events:

- By overriding the **onTouchEvent()** method usually for the whole Activity screen
- By registering an **onTouchListener** and the implementation of the onTouch() callback method

BEFORE YOU BEGIN: Download the following application project as a starter application.

ActivityA_Touch Board_Starter.zip

The above project contains a few TextViews placed on *activity_main* to give you a good start.

A.1 Override onTouchEvent

onTouchEvent method is available from MainActivity's parent class. Once overridden, the initial method block will be created with the following code block.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    return super.onTouchEvent(event);
}
```

A.2 Detect type of Touch Event

Information about Touch Event is inside the MotionEvent variable supplied as an input parameter of the *onTouchEvent* method. To run specific code based on the type of Touch Event, we can write conditional code blocks as shown below.

```
/**
```

```java
* Called when a touch screen event was not handled by any of the views inside of the activity.
*
* @param event The touch screen event being processed.
*
* @return must return true if event was consumed by our custom programming logic.
*/
@Override
public boolean onTouchEvent(MotionEvent event) {

    // get the type of Motion Event detected which is represented by a pre-defined integer value
    int action = event.getAction();

    // compare the detected event type against pre-defined values
    if (action == MotionEvent.ACTION_DOWN){
        tvEventType.setText("ACTION_DOWN");
    } else if (action == MotionEvent.ACTION_UP){
        tvEventType.setText("ACTION_UP");
    } else if (action == MotionEvent.ACTION_MOVE){
        tvEventType.setText("ACTION_MOVE");
    }
    return true;
}
```

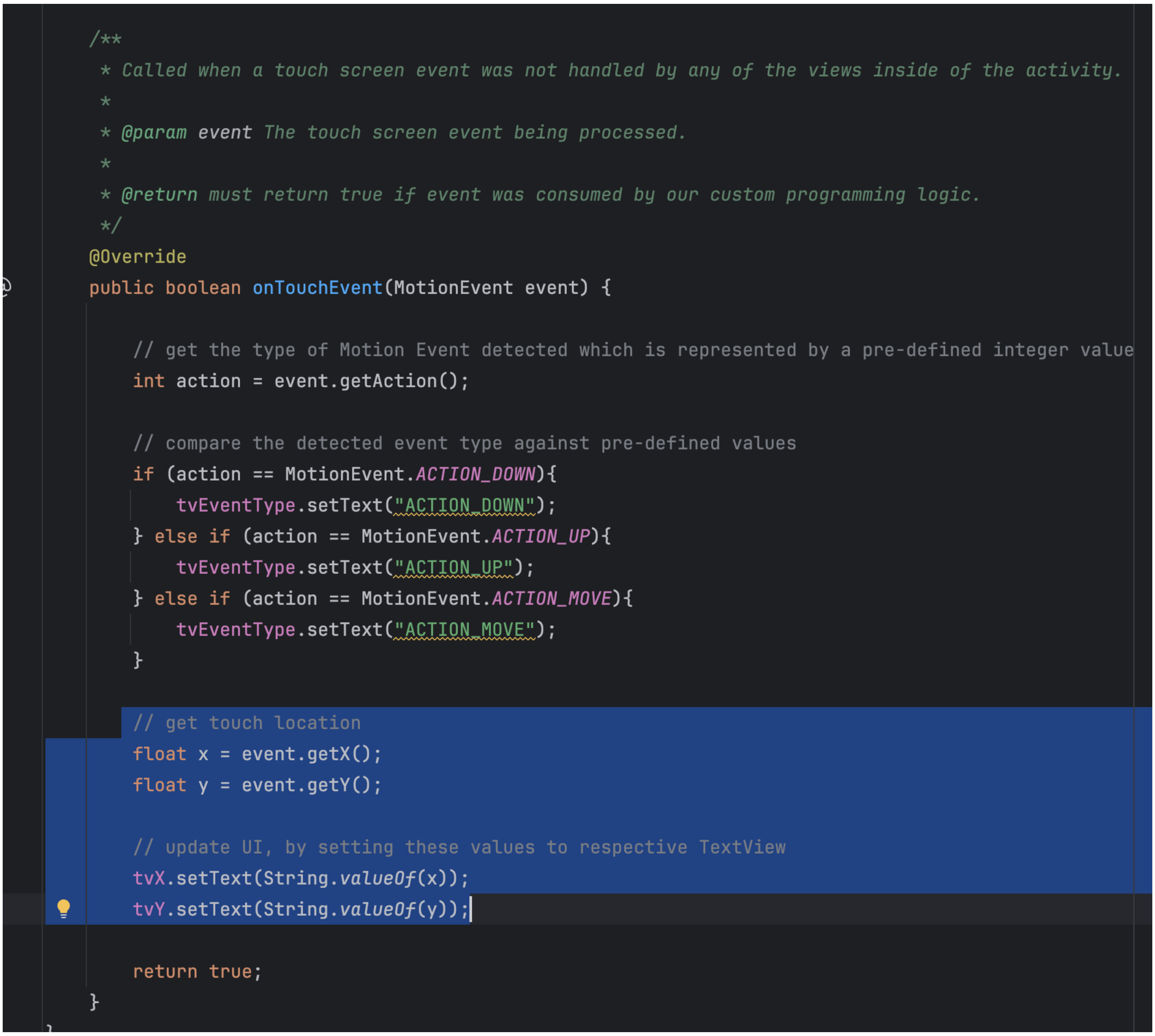
⚠ After adding the method above, test your app and check with lab tutors if you run into any issues.

A.3 Extract the position of *MotionEvent*

Information about where the touch event occurred is also available from the *event* parameter, you can extract that using the *getX()* & *getY()* methods.

```java
// get touch location
float x = event.getX();
float y = event.getY();

// update UI, by setting these values to respective TextView
tvX.setText(String.valueOf(x));
tvY.setText(String.valueOf(y));
```

After implementing the above code-block the state of onTouchEvent should be as shown below.

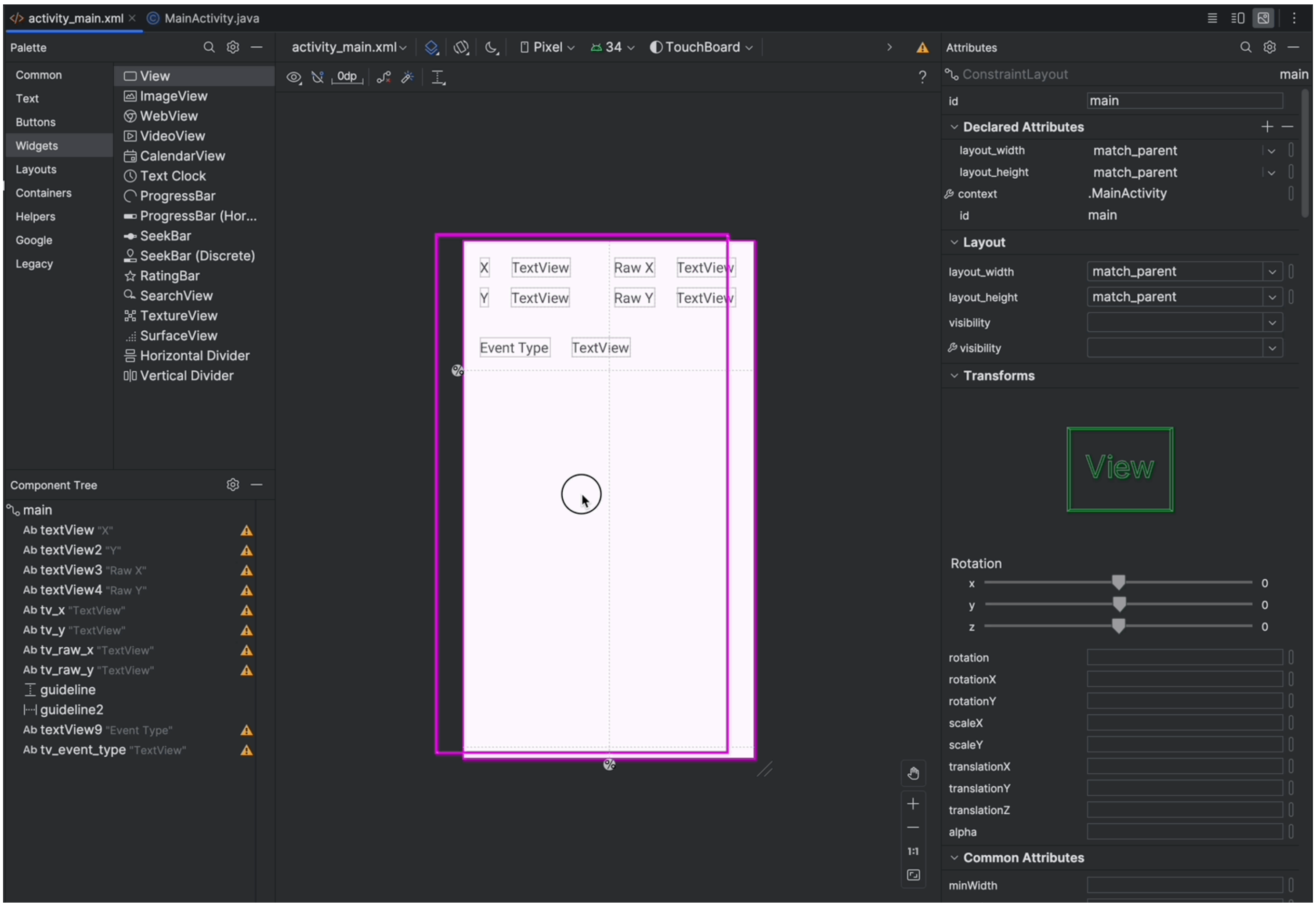


```
    /**
     * Called when a touch screen event was not handled by any of the views inside of the activity.
     *
     * @param event The touch screen event being processed.
     *
     * @return must return true if event was consumed by our custom programming logic.
     */
    @Override
    public boolean onTouchEvent(MotionEvent event) {

        // get the type of Motion Event detected which is represented by a pre-defined integer value
        int action = event.getAction();

        // compare the detected event type against pre-defined values
        if (action == MotionEvent.ACTION_DOWN){
            tvEventType.setText("ACTION_DOWN");
        } else if (action == MotionEvent.ACTION_UP){
            tvEventType.setText("ACTION_UP");
        } else if (action == MotionEvent.ACTION_MOVE){
            tvEventType.setText("ACTION_MOVE");
        }

        // get touch location
        float x = event.getX();
        float y = event.getY();

        // update UI, by setting these values to respective TextView
        tvX.setText(String.valueOf(x));
        tvY.setText(String.valueOf(y));

        return true;
    }
```

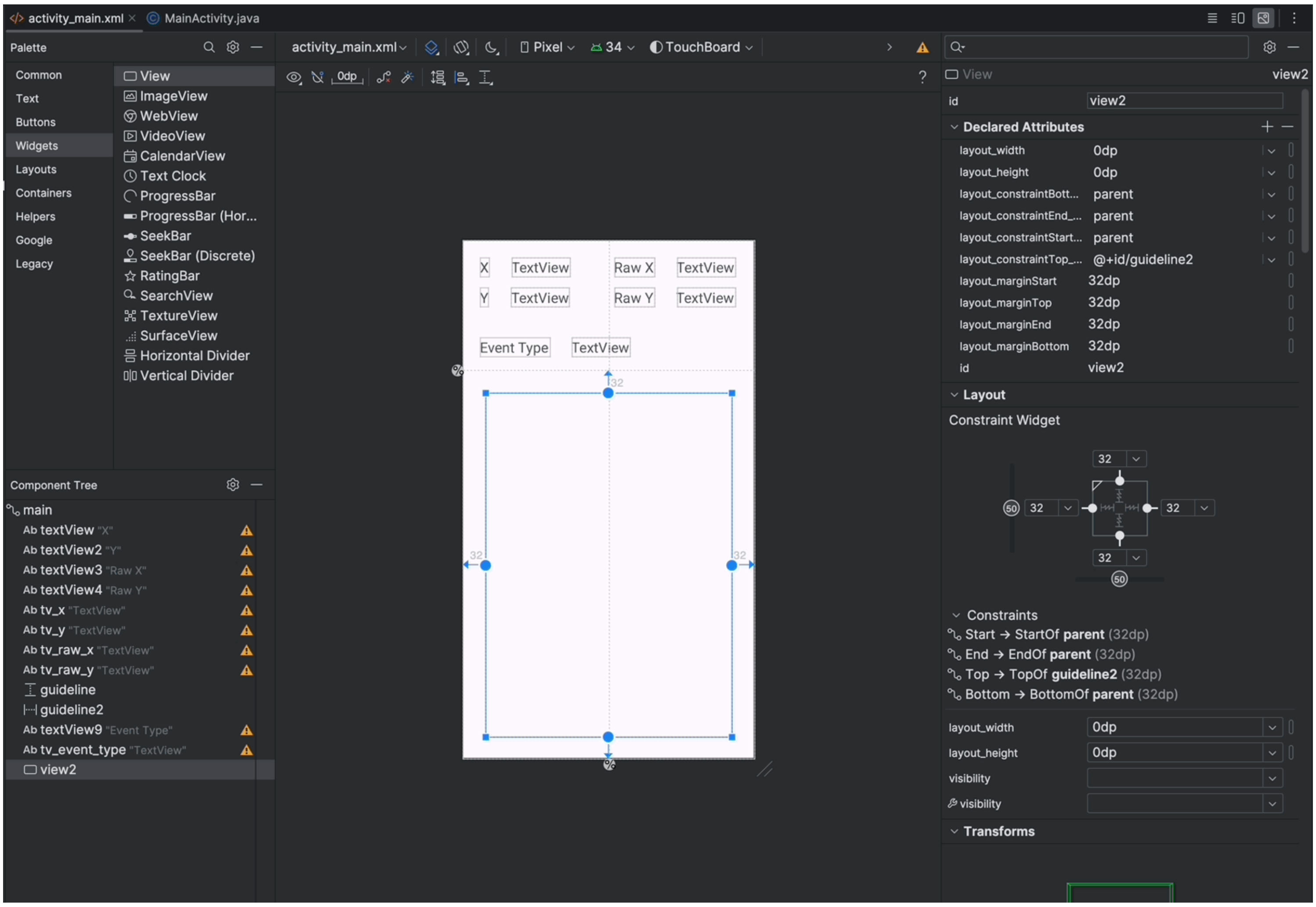
Run the app and try tapping anywhere on there screen, drag your finger around on the screen and monitor how X & Y values are changing & the type of motion event.

A.4 Add new *View* - Touchpad

So far we captured action events on any region of the MainActivity, how about capturing events within restricted areas like a touchpad, drawing canvas, or any custom view element with pre-defined boundaries? Let's create a new View element and change its background to blue, follow the steps below.



Now set the ID of this view and change the background colour to blue (or your favourite colour).



View view2

id	view2

Declared Attributes

layout_width	0dp
layout_height	0dp
layout_constraintBott...	parent
layout_constraintEnd_...	parent
layout_constraintStart...	parent
layout_constraintTop_...	@+id/guideline2
layout_marginStart	32dp
layout_marginTop	32dp
layout_marginEnd	32dp
layout_marginBottom	32dp
id	view2

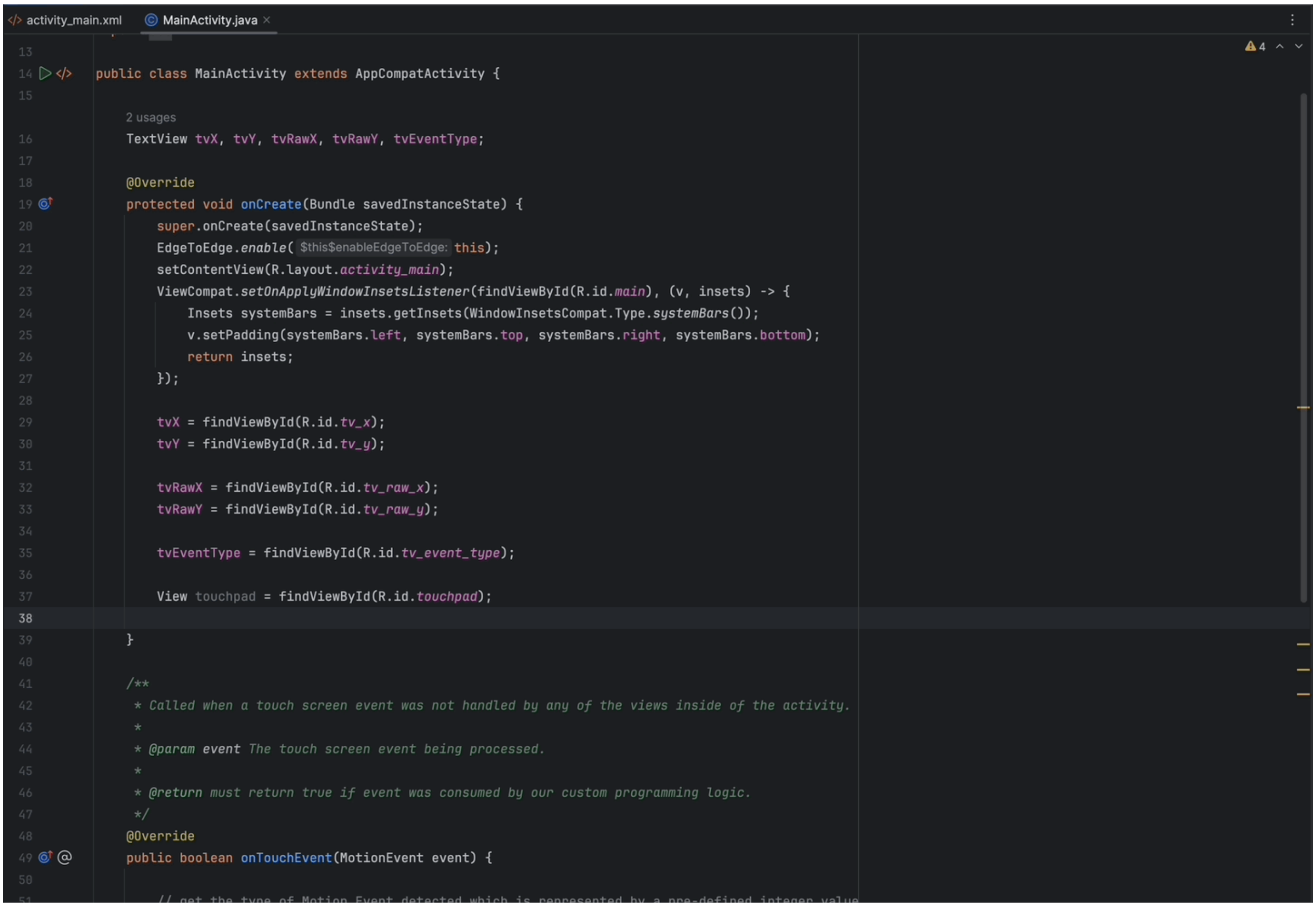
Constraints
Start → StartOf **parent** (32dp)
End → EndOf **parent** (32dp)
Top → TopOf **guideline2** (32dp)
Bottom → BottomOf **parent** (32dp)

layout_width	0dp
layout_height	0dp
visibility	
visibility	

A.5 Register Touch Event Listener

To capture events within the touchpad area, we need to register for touch events using the listener as described using the steps below.

```
// get reference to the View element using the Id we specified above
View touchpad = findViewById(R.id.touchpad);
```



```java
public class MainActivity extends AppCompatActivity {

    2 usages
    TextView tvX, tvY, tvRawX, tvRawY, tvEventType;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        tvX = findViewById(R.id.tv_x);
        tvY = findViewById(R.id.tv_y);

        tvRawX = findViewById(R.id.tv_raw_x);
        tvRawY = findViewById(R.id.tv_raw_y);

        tvEventType = findViewById(R.id.tv_event_type);

        View touchpad = findViewById(R.id.touchpad);

    }

    /**
     * Called when a touch screen event was not handled by any of the views inside of the activity.
     *
     * @param event The touch screen event being processed.
     *
     * @return must return true if event was consumed by our custom programming logic.
     */
    @Override
    public boolean onTouchEvent(MotionEvent event) {

        // get the type of Motion Event detected which is represented by a pre-defined integer value
```

A.6 Move code from *onTouchEvent* to *onTouch*

Move the code of the *onTouchEvent* method to the *onTouch* method of the touchpad's touch event listener as shown below.

Run the app and test the same functionalities you tested before keeping in mind now only the TouchPad should detect motion events and any motion outside the touchpad if started is ignored.

A.7 Implement code to capture RawX & RawY values

Since, we are only listening for touch events on the touchpad area, if we need to receive coordinates with respect to the device screen we can use the **getRawX** & **getRawY** methods.

```
// get raw x & y touch coordinates
float rawX = event.getRawX();
float rawY = event.getRawY();

// update UI, by setting these values to respective TextView
tvRawX.setText(String.valueOf(rawX));
tvRawY.setText(String.valueOf(rawY));
```

That's all for this activity, think of this as a foundation on which different View elements are built, for example:

- A button click comprises a DOWN & UP event within the button's boundaries.
- Scroll or swipe gesture is a combination of DOWN, MOVE & UP within the RecyclerView pane or

ScrollView.

B. Gesture Detectors

B.0 Gesture Detectors

Gesture is basically a meanigful sequence of touch events and Android provides us with common gestures ready to use from GestureDetector class. This activity is a step-by-step guide of how to implement gesture detectors from scratch.

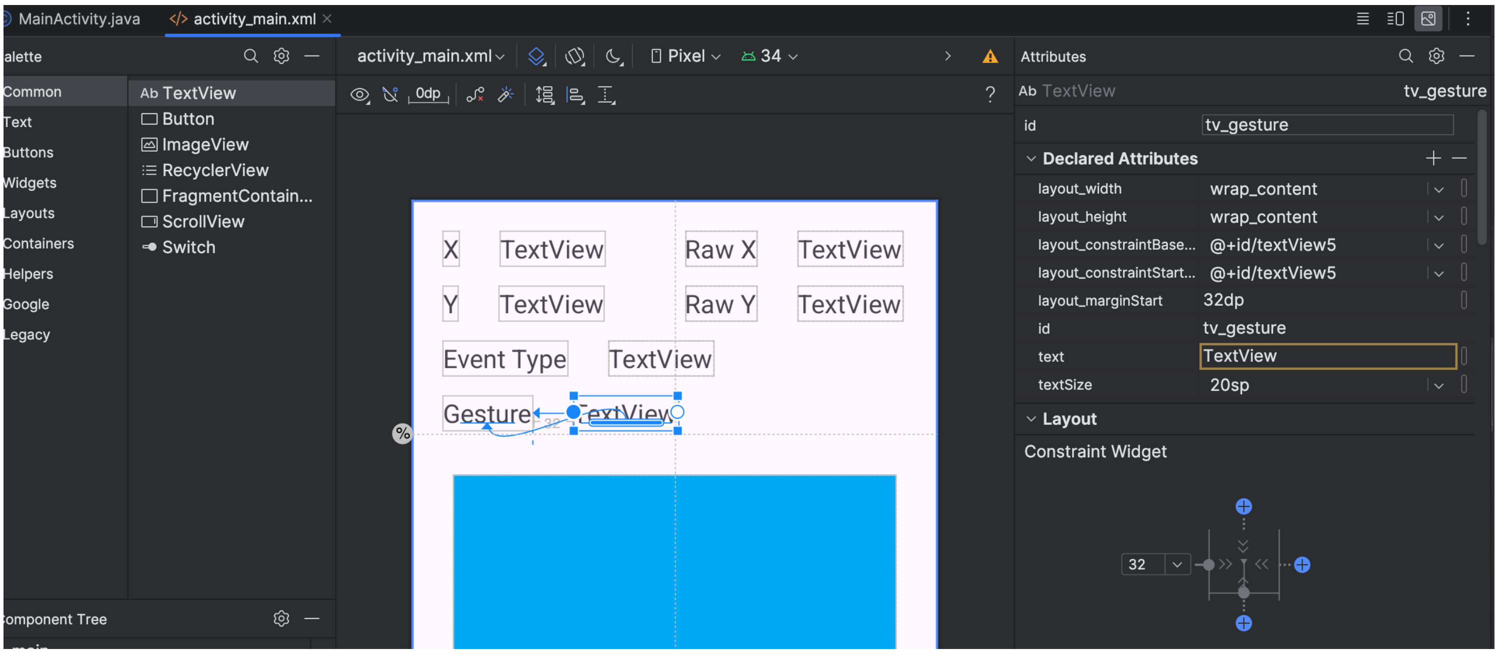
BEFORE YOU BEGIN: Implement Activity A, and use the same project to implement the following.

Learning material:

https://edstem.org/au/courses/14448/lessons/49208/slides/357395

https://edstem.org/au/courses/14448/lessons/49208/slides/357979

B.1 Add new TextViews to display current Gesture



Attributes

TextView tv_gesture

id	tv_gesture

Declared Attributes

layout_width	wrap_content
layout_height	wrap_content
layout_constraintBase...	@+id/textView5
layout_constraintStart...	@+id/textView5
layout_marginStart	32dp
id	tv_gesture
text	TextView
textSize	20sp

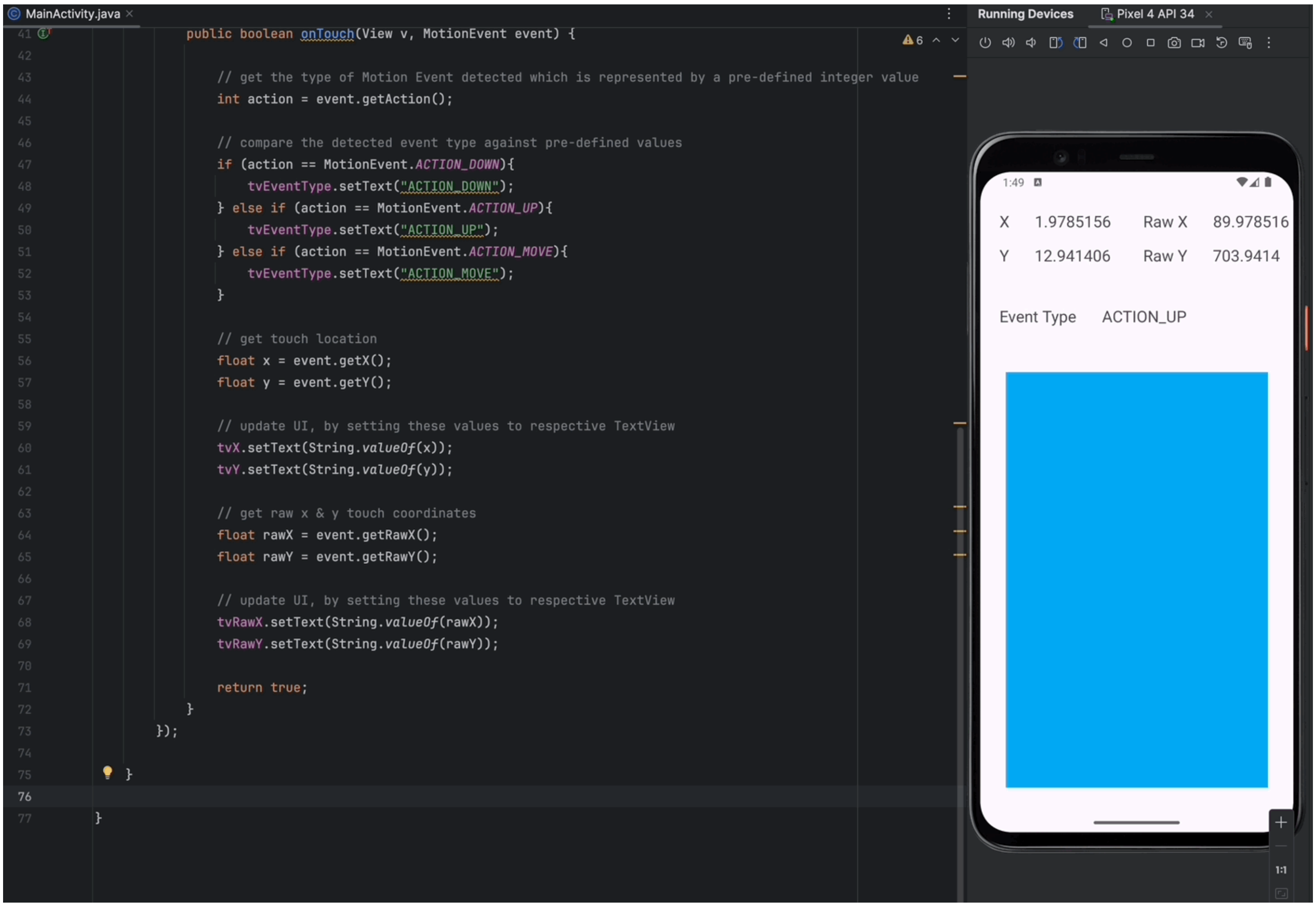
Layout

```
// declare as a class variable
TextView tvGesture;
```

```
// initialise in onCreate method
tvGesture = findViewById(R.id.tv_gesture);
```

B.2 Create a new inner class *CustomGestureDetector*

Inside MainActivity create a new class *CustomGestureDetector* and inherit GestureDetector.SimpleOnGestureListener class.


MainActivity.java

Running Devices Pixel 4 API 34

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

                // get the type of Motion Event detected which is represented by a pre-defined integer value
                int action = event.getAction();

                // compare the detected event type against pre-defined values
                if (action == MotionEvent.ACTION_DOWN){
                    tvEventType.setText("ACTION_DOWN");
                } else if (action == MotionEvent.ACTION_UP){
                    tvEventType.setText("ACTION_UP");
                } else if (action == MotionEvent.ACTION_MOVE){
                    tvEventType.setText("ACTION_MOVE");
                }

                // get touch location
                float x = event.getX();
                float y = event.getY();

                // update UI, by setting these values to respective TextView
                tvX.setText(String.valueOf(x));
                tvY.setText(String.valueOf(y));

                // get raw x & y touch coordinates
                float rawX = event.getRawX();
                float rawY = event.getRawY();

                // update UI, by setting these values to respective TextView
                tvRawX.setText(String.valueOf(rawX));
                tvRawY.setText(String.valueOf(rawY));

                return true;
            }
        });

    }

}
```

```
X    1.9785156    Raw X    89.978516
Y    12.941406    Raw Y    703.9414

Event Type    ACTION_UP
```

```
/**
 * A convenience class to extend when you only want to listen for a subset of all the gestures.
 */
class CustomGestureDetector extends GestureDetector.SimpleOnGestureListener{


}
```

B.3 Add class variables - MainActivity

Intordure these two new class variables inside MainActivity. These variables will help us intialise Gesture detector for the touchpad.

```
// help detect basic gestures like scroll, single tap, double tap, etc
```

```
private GestureDetectorCompat mDetector;

// help detect multi touch gesture like pinch-in, pinch-out specifically used for zoom functionalit
private ScaleGestureDetector mScaleDetector;
```

import android.view.ScaleGestureDetector;
import androidx.core.view.GestureDetectorCompat;

B.4 Register custom detector class as GestureDetector

Registering for gesture detector is one time activity and can be placed inside **onCreate** method. This is equivalent to setting onClick listener on a button, however in this case we are setting listener as *CustomGestureDetector*.

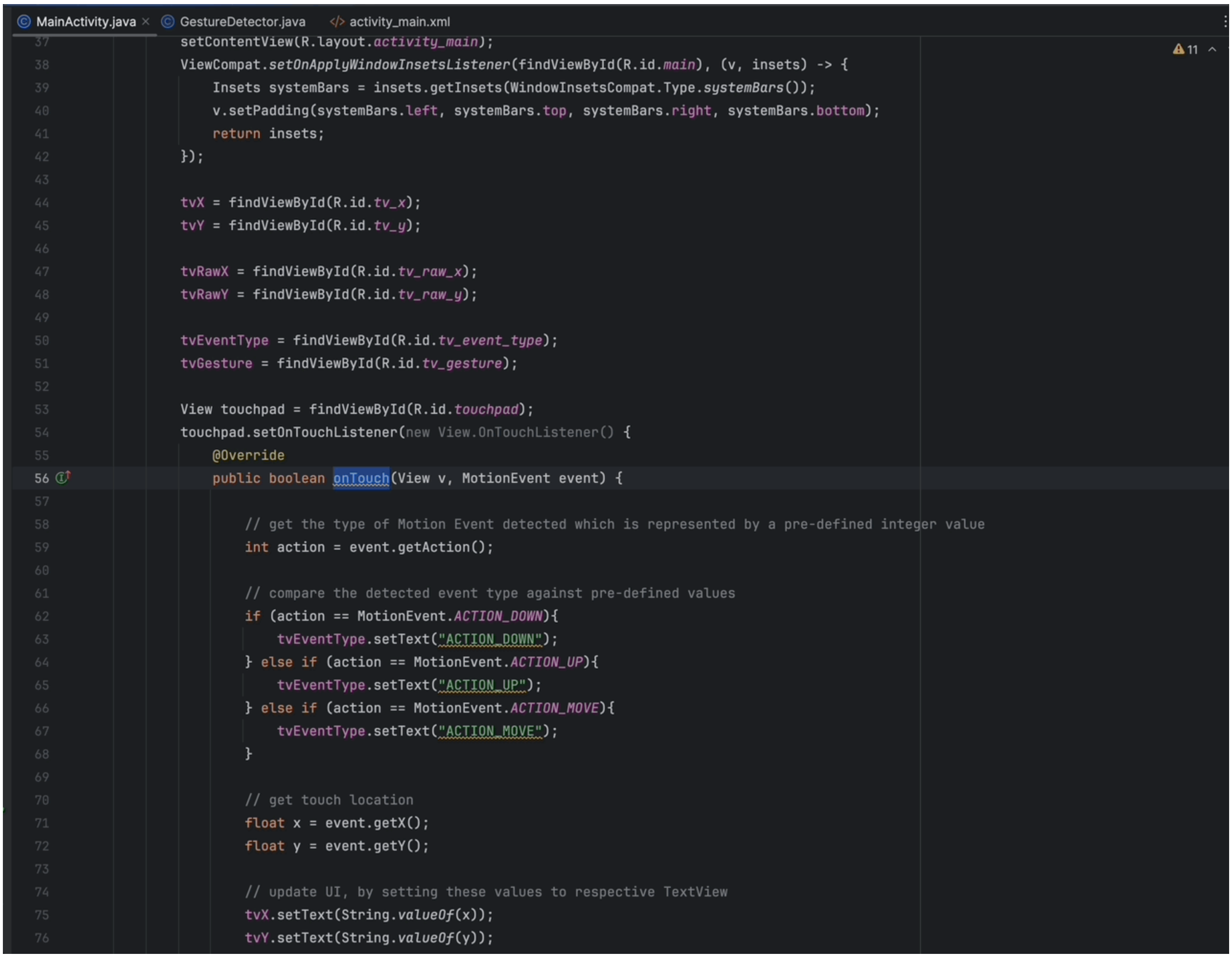
```
// initialise new instance of CustomGestureDetector class
CustomGestureDetector customGestureDetector = new CustomGestureDetector();

// register GestureDetector and set listener as CustomGestureDetector
mDetector = new GestureDetectorCompat(this, customGestureDetector);
```

Due to above, any gesture detected by GestureDetectorCompat will be passed on to CustomGestureDetector and there we can implement our custom implementation of listening to one of the pre-defined gestures.

B.5 Pass basic touch events to GestureDetectorCompat

This is important step, as we need to pass stream of touch events to GestureDetectorCompat, so it can interpret the sequence of events and convert it into gestures.

```
// pass stream of events to GestureDetectorCompat, which translate events into gestures
mDetector.onTouchEvent(event);
```



```java
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        tvX = findViewById(R.id.tv_x);
        tvY = findViewById(R.id.tv_y);

        tvRawX = findViewById(R.id.tv_raw_x);
        tvRawY = findViewById(R.id.tv_raw_y);

        tvEventType = findViewById(R.id.tv_event_type);
        tvGesture = findViewById(R.id.tv_gesture);

        View touchpad = findViewById(R.id.touchpad);
        touchpad.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {

                // get the type of Motion Event detected which is represented by a pre-defined integer value
                int action = event.getAction();

                // compare the detected event type against pre-defined values
                if (action == MotionEvent.ACTION_DOWN){
                    tvEventType.setText("ACTION_DOWN");
                } else if (action == MotionEvent.ACTION_UP){
                    tvEventType.setText("ACTION_UP");
                } else if (action == MotionEvent.ACTION_MOVE){
                    tvEventType.setText("ACTION_MOVE");
                }

                // get touch location
                float x = event.getX();
                float y = event.getY();

                // update UI, by setting these values to respective TextView
                tvX.setText(String.valueOf(x));
                tvY.setText(String.valueOf(y));
```

B.6 Override common gestures

Override following gesture callback and set the text of "*tv_gesture*" with the name of current gesture.

- onDown
- onSingleTapConfirmed
- onScroll
- onFling
- onLongPress
- onDoubleTap

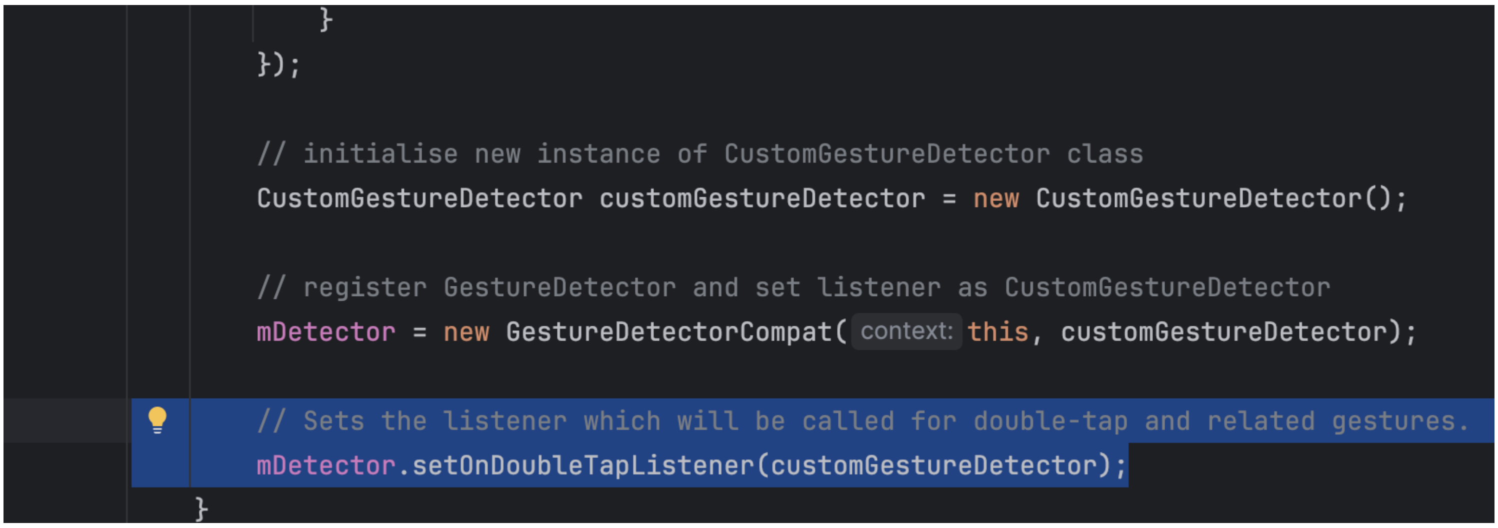
```java
/**
* Custom class inheriting a convenience class to extend when you only want to listen for a subset o
*/
class CustomGestureDetector extends GestureDetector.SimpleOnGestureListener{

    @Override
    public boolean onDown(@NonNull MotionEvent e) {
        tvGesture.setText("onDown");
        return super.onDown(e);
    }

    @Override
    public boolean onSingleTapConfirmed(@NonNull MotionEvent e) {
        tvGesture.setText("onSingleTapConfirmed");
        return super.onSingleTapConfirmed(e);
    }

    @Override
    public boolean onScroll(@Nullable MotionEvent e1, @NonNull MotionEvent e2, float distanceX, flo
        tvGesture.setText("onScroll");
        return super.onScroll(e1, e2, distanceX, distanceY);
    }

    @Override
    public boolean onFling(@Nullable MotionEvent e1, @NonNull MotionEvent e2, float velocityX, floa
        tvGesture.setText("onFling");
        return super.onFling(e1, e2, velocityX, velocityY);
    }

    @Override
    public void onLongPress(@NonNull MotionEvent e) {
        tvGesture.setText("onLongPress");
        super.onLongPress(e);
    }

    @Override
    public boolean onDoubleTap(@NonNull MotionEvent e) {
        tvGesture.setText("onDoubleTap");
        return super.onDoubleTap(e);
    }

}
```

Try running the app and test out all the gestures, see if you could identify which callback method is not getting registered.

B.7 Double Tap Listener

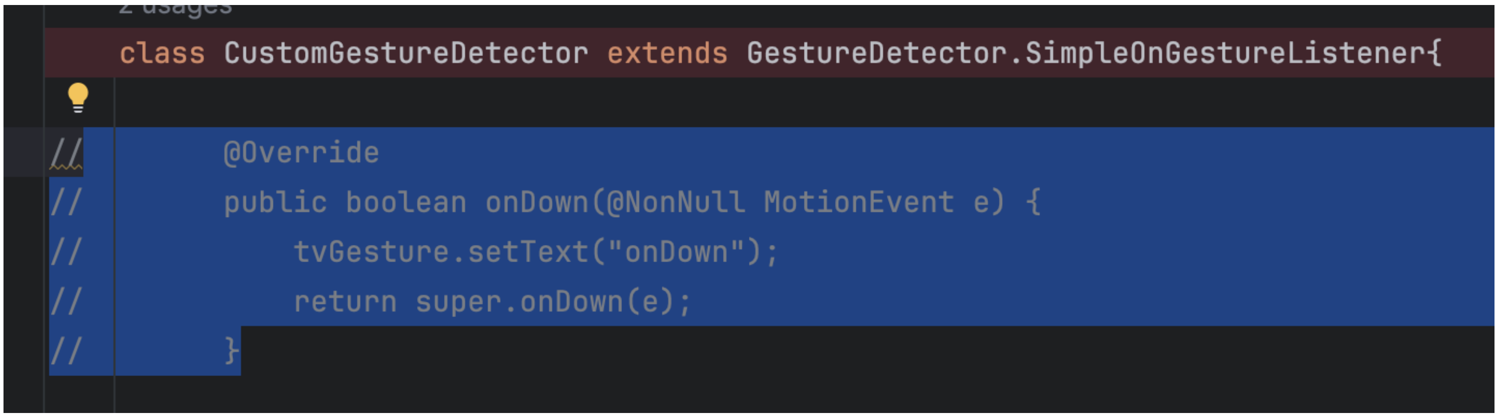
Till previous step, all of the gesture callbacks would work fine except onDoubleTap, this is because double tap gestures require two additional changes

Set double tap listener



```
            }
        });

        // initialise new instance of CustomGestureDetector class
        CustomGestureDetector customGestureDetector = new CustomGestureDetector();

        // register GestureDetector and set listener as CustomGestureDetector
        mDetector = new GestureDetectorCompat( context: this, customGestureDetector);

        // Sets the listener which will be called for double-tap and related gestures.
        mDetector.setOnDoubleTapListener(customGestureDetector);
    }
```

Comment out onDown gesture



```
2 usages
class CustomGestureDetector extends GestureDetector.SimpleOnGestureListener{

//        @Override
//        public boolean onDown(@NonNull MotionEvent e) {
//            tvGesture.setText("onDown");
//            return super.onDown(e);
//        }
```

This is because onDown gets executed multiple times in a Double tap gesture.

B.8 Scale gesture detector

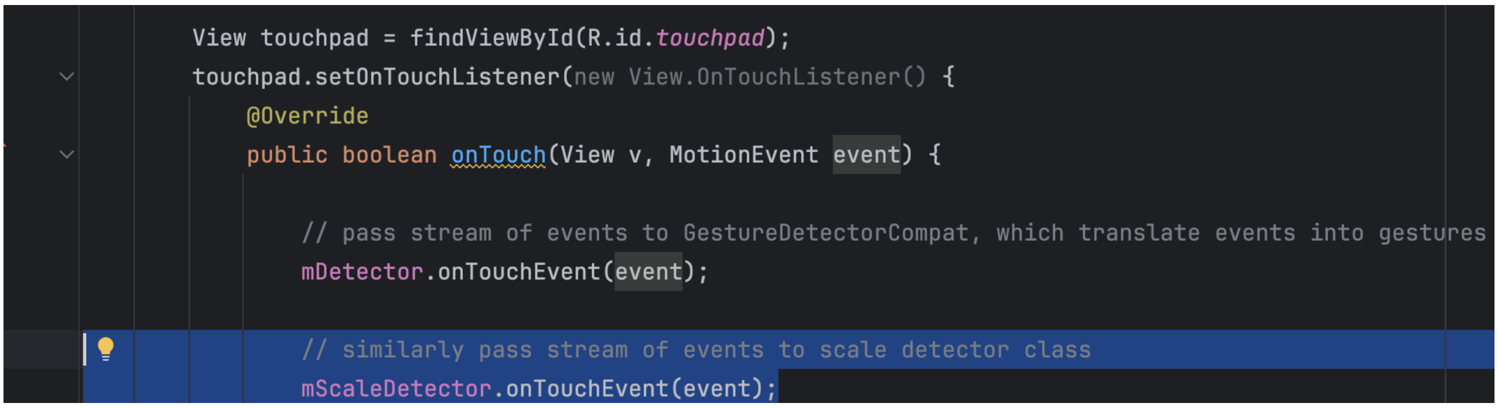
Similar concept for scale gestures as well, however, it would require implementation of SimpleScaleGestureDetector.

```
/**
 * A custom Java class which inherits a convenience class to listen to scaling-related events.
 */
class CustomScaleGestureDetector extends ScaleGestureDetector.SimpleOnScaleGestureListener{

    @Override
    public boolean onScale(@NonNull ScaleGestureDetector detector) {
        tvGesture.setText("onScale");
        return super.onScale(detector);
    }

    @Override
    public boolean onScaleBegin(@NonNull ScaleGestureDetector detector) {
        tvGesture.setText("onScaleBegin");
        return super.onScaleBegin(detector);
    }

    @Override
    public void onScaleEnd(@NonNull ScaleGestureDetector detector) {
        tvGesture.setText("onScaleEnd");
        super.onScaleEnd(detector);
    }
}
```

Inside onCreate inialise ScaleGestureDetector class variable (added as part of A3).

```
// onCreate: register CustomScaleGestureDetector class to listen scale gestures
mScaleDetector = new ScaleGestureDetector(this, new CustomScaleGestureDetector());
```

Finally, pass basic events to ScaleGestureDetector like we previously did for SimpleGestureDetector.



```
View touchpad = findViewById(R.id.touchpad);
touchpad.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        // pass stream of events to GestureDetectorCompat, which translate events into gestures
        mDetector.onTouchEvent(event);

        // similarly pass stream of events to scale detector class
        mScaleDetector.onTouchEvent(event);
```

Well done, you are now fully equipped to handle simple and multi touch gestures!

C. Draw Custom Shapes

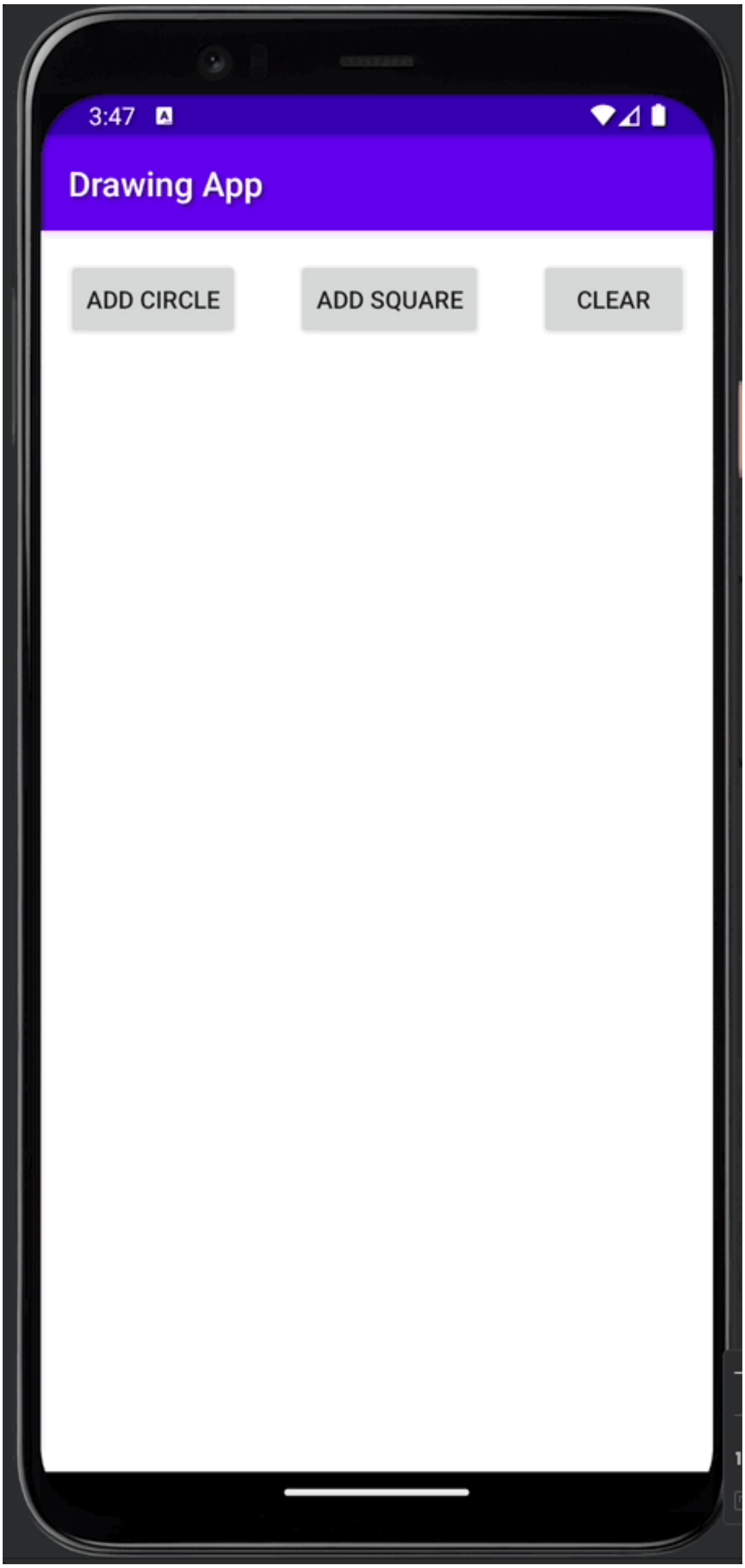
C.0 CustomView - Draw Shapes

BEFORE YOU BEGIN: Download the following application project as a starter application.

Drawing App_Start.zip

The app provided above contains following features:

- Demostrates an example of CustomView (a new Java class which represents custom UI element).
- CustomView inherits ***View*** class and contains logic to draw custom shapes
- "Add Circle" button adds a new circle at random location on the screen, radius 100.
- "Add Square" button adds a new square at random location on the screen, height and width 100.
- "Clear" button resets the view and delete all drawn shapes from CustomView


3:47
Drawing App
ADD CIRCLE
ADD SQUARE
CLEAR

Using gesture detectors, implement the following.

C.1 Single Tap Confirmed Gesture - Draw Circle

If scroll gesture is detected, draw a circle at current scroll location.

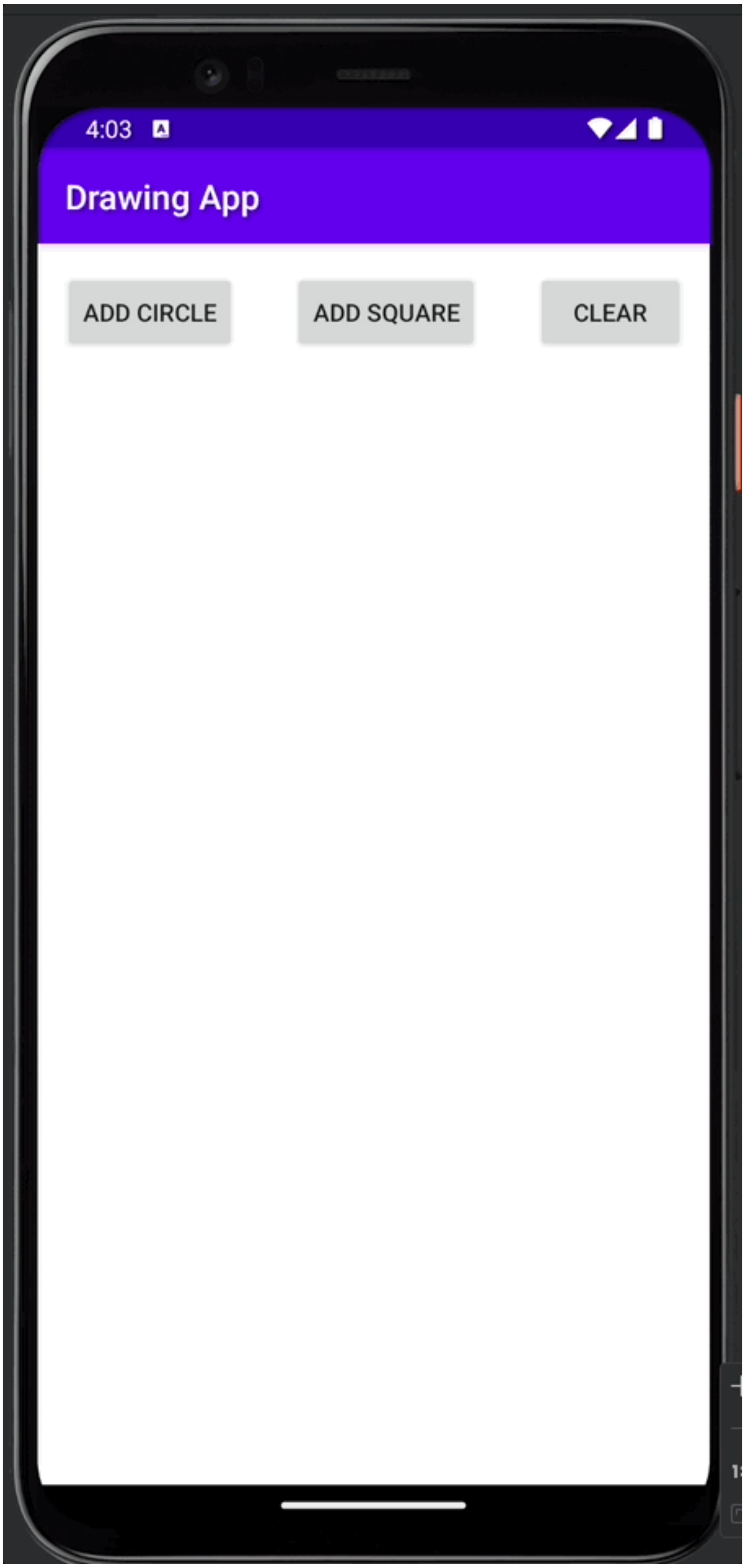
```
@Override
public boolean onScroll(@Nullable MotionEvent e1, @NonNull MotionEvent e2, float distanceX, float d
    view.addShape(new Circle((int)e2.getX(),(int)e2.getY(), 100));
    return super.onScroll(e1, e2, distanceX, distanceY);
}
```

C.2 Double Tap Gesture - Draw Square

If Double tap gesture is detected, draw a square at double tap location.

```
@Override
public boolean onDoubleTap(@NonNull MotionEvent e) {
    view.addShape(new Rectangle((int)e.getX(),(int)e.getY(), 100, 100));
    return super.onDoubleTap(e);
}
```

Here is expected output for your reference, in case you encounter any issues check with you lab tutors.


4:03
Drawing App
ADD CIRCLE
ADD SQUARE
CLEAR