

# MULTI-TOUCH & GESTURES

# TODAY'S TOPICS

MOTIONEVENTS

TOUCH HANDLING

GESTURES

# MOTIONEVENT

REPRESENTS A MOVEMENT IN AN INPUT  
DEVICE READING

PEN, TRACKBALL, MOUSE, FINGER



# MOTIONEVENT

## ACTION CODE

STATE CHANGE THAT OCCURRED

## ACTION VALUES

POSITION AND MOVEMENT PROPERTIES,  
SUCH AS TIME, SOURCE, LOCATION,  
PRESSURE, AND MORE

THIS LESSON FOCUSES ON TOUCH  
EVENTS READ FROM A TOUCH SCREEN

# MULTITOUCH

MULTITOUCH SCREENS EMIT ONE MOVEMENT  
TRACE PER TOUCH SOURCE

INDIVIDUAL TOUCH SOURCES ARE CALLED  
POINTERS

# MULTITOUCH

EACH POINTER HAS A UNIQUE ID FOR AS LONG AS IT IS ACTIVE

MotionEventS CAN REFER TO MULTIPLE POINTERS

EACH POINTER HAS AN INDEX WITHIN THE EVENT, BUT THAT INDEX MAY NOT BE STABLE OVER TIME

# SOME MotionEvent ACTIONS

ACTION\_DOWN

ACTION\_POINTER\_DOWN

ACTION\_POINTER\_UP

ACTION\_MOVE

ACTION\_UP

ACTION\_CANCEL



# CONSISTENCY GUARANTEES

FOR TOUCH EVENTS, ANDROID TRIES TO  
GUARANTEE THAT TOUCHES

GO DOWN ONE AT A TIME

MOVE AS A GROUP

COME UP ONE AT A TIME OR ARE CANCELLED

APPLICATIONS SHOULD BE TOLERANT TO  
INCONSISTENCY

# MOTIONEVENT METHODS

getActionMasked()

getActionIndex()

getPointerId(int pointerIndex)

getPointerCount()

getX(int pointerIndex)

getY(int pointerIndex)

findPointerIndex (int pointerId)

# HANDLING TOUCH EVENTS ON A VIEW

THE VIEW BEING TOUCHED RECEIVES

`View.onTouchEvent(MotionEvent event)`

`onTouchEvent()` SHOULD RETURN TRUE IF

THE `MotionEvent` HAS BEEN CONSUMED;

FALSE OTHERWISE

# HANDLING TOUCH EVENTS WITH A LISTENER

View.OnTouchListener DEFINES TOUCH  
EVENT CALLBACK METHODS

View.setOnTouchListener() REGISTERS  
LISTENER FOR TOUCH CALLBACKS

# HANDLING TOUCH EVENTS WITH A LISTENER

onTouch() CALLED WHEN A TOUCH EVENT, SUCH AS PRESSING, RELEASING OR DRAGGING, OCCURS

onTouch() CALLED BEFORE THE EVENT IS DELIVERED TO THE TOUCHED VIEW

SHOULD RETURN TRUE IF IT HAS CONSUMED THE EVENT; FALSE OTHERWISE

# HANDLING MULTIPLE TOUCH EVENTS

MULTIPLE TOUCHES COMBINED TO FORM A  
MORE COMPLEX GESTURE

IDENTIFY & PROCESS COMBINATIONS OF  
TOUCHES,

FOR EXAMPLE, A DOUBLE TAP

ACTION\_DOWN, ACTION\_UP,  
ACTION\_DOWN, ACTION\_UP IN QUICK  
SUCCESSION

# MULTI-TOUCH HANDLING



# MULTI-TOUCH HANDLING

	Action	IDs
➔ #1 touch ➔	ACTION_DOWN	0
	ACTION_MOVE ...	0
#2 touch ➔	ACTION_POINTER_DOWN	1
	ACTION_MOVE ...	0,1
#1 lift ➔	ACTION_POINTER_UP	0
#2 lift ➔	ACTION_UP	1

# MULTI-TOUCH HANDLING

# MULTI-TOUCH HANDLING

	Action	ID
➔ #1 touch ➔	ACTION_DOWN	0
	ACTION_MOVE ...	0
#2 touch ➔	ACTION_POINTER_DOWN	1
	ACTION_MOVE ...	0,1
#2 lift ➔	ACTION_POINTER_UP	1
#1 lift ➔	ACTION_UP	0

# MULTI-TOUCH HANDLING

# MULTI-TOUCH HANDLING

➔ #1 touch →

#2 touch →

#3 touch →

#2 lift →

#1 lift →

#3 lift →

Action	ID
ACTION_DOWN	0
ACTION_POINTER_DOWN	1
ACTION_POINTER_DOWN	2
ACTION_MOVE	0,1,2
ACTION_POINTER_UP	1
ACTION_POINTER_UP	0
ACTION_UP	2

# TOUCHINDICATE TOUCHLOCATION

APPLICATION DRAWS A CIRCLE WHEREVER THE  
USERS TOUCHES THE SCREEN

CIRCLE'S COLOR IS RANDOMLY SELECTED

REDRAWS CIRCLES WHEN USER DRAGS  
FINGER ACROSS THE SCREEN

# TOUCHINDICATE TOUCHLOCATION

THE SIZE OF THE CIRCLES ARE PROPORTIONAL  
TO THE NUMBER OF CURRENTLY ACTIVE  
TOUCHES





# TOUCHINDICATETOUCHLOCATION

```
public class IndicateTouchLocationActivity extends Activity {  
  
    private static final int MIN_DXDY = 2;  
  
    // Assume no more than 20 simultaneous touches  
    final private static int MAX_TOUCHES = 20;  
  
    // Pool of MarkerViews  
    final private static LinkedList<MarkerView> mInactiveMarkers = new LinkedList<MarkerView>();  
  
    // Set of MarkerViews currently visible on the display  
    @SuppressWarnings("UseSparseArrays")  
    final private static Map<Integer, MarkerView> mActiveMarkers = new HashMap<Integer, MarkerView>();  
  
    protected static final String TAG = "IndicateTouchLocationActivity";  
  
    private FrameLayout mFrame;
```

# TOUCHINDICATETOUCHLOCATION

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mFrame = (FrameLayout) findViewById(R.id.frame);

    // Initialize pool of View.
    initViews();
}
```

# TOUCHINDICATE TOUCHLOCATION

```
// Create and set on touch listener
mFrame.setOnTouchListener(new OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {

        switch (event.getActionMasked()) {

            // Show new MarkerView

            case MotionEvent.ACTION_DOWN:
            case MotionEvent.ACTION_POINTER_DOWN: {

                int pointerIndex = event.getActionIndex();
                int pointerID = event.getPointerId(pointerIndex);

                MarkerView marker = mInactiveMarkers.remove();

                if (null != marker) {
                    mActiveMarkers.put(pointerID, marker);

                    marker.setXLoc(event.getX(pointerIndex));
                    marker.setYLoc(event.getY(pointerIndex));

                    updateTouches(mActiveMarkers.size());

                    mFrame.addView(marker);
                }
                break;
            }
        }
    }
});
```

# TOUCHINDICATE TOUCHLOCATION

```
// Remove one MarkerView

case MotionEvent.ACTION_UP:
case MotionEvent.ACTION_POINTER_UP: {

    int pointerIndex = event.getActionIndex();
    int pointerID = event.getPointerId(pointerIndex);

    MarkerView marker = mActiveMarkers.remove(pointerID);

    if (null != marker) {

        mInactiveMarkers.add(marker);

        updateTouches(mActiveMarkers.size());

        mFrame.removeView(marker);
    }
    break;
}
```

# TOUCHINDICATETOUCHLOCATION

```
// Move all currently active MarkerViews

case MotionEvent.ACTION_MOVE: {

    for (int idx = 0; idx < event.getPointerCount(); idx++) {

        int ID = event.getPointerId(idx);

        MarkerView marker = mActiveMarkers.get(ID);
        if (null != marker) {

            // Redraw only if finger has travel ed a minimum distance
            if (Math.abs(marker.getXLoc() - event.getX(idx)) > MIN_DXDY
                || Math.abs(marker.getYLoc()
                    - event.getY(idx)) > MIN_DXDY) {

                // Set new location

                marker.setXLoc(event.getX(idx));
                marker.setYLoc(event.getY(idx));

                // Request re-draw
                marker.invalidate();

            }
        }
    }

    break;
}
```

# TOUCHINDICATETOUCHLOCATION

```
        default:
            Log.i(TAG, "unhandled action");
        }

        return true;
    }

    // update number of touches on each active MarkerView
    private void updateTouches(int numActive) {
        for (MarkerView marker : mActiveMarkers.values()) {
            marker.setTouches(numActive);
        }
    }
    });
}

private void initView() {
    for (int idx = 0; idx < MAX_TOUCHES; idx++) {
        mInactiveMarkers.add(new MarkerView(this, -1, -1));
    }
}
```



# TOUCHINDICATETOUCHLOCATION

```
private class MarkerView extends View {  
    private float mX, mY;  
    final static private int MAX_SIZE = 400;  
    private int mTouches = 0;  
    final private Paint mPaint = new Paint();  
  
    public MarkerView(Context context, float x, float y) {  
        super(context);  
        mX = x;  
        mY = y;  
        mPaint.setStyle(Style.FILL);  
  
        Random rnd = new Random();  
        mPaint.setARGB(255, rnd.nextInt(256), rnd.nextInt(256),  
            rnd.nextInt(256));  
    }  
}
```

# TOUCHINDICATETOUCHLOCATION

```
float getXLoc() {  
    return mX;  
}  
  
void setXLoc(float x) {  
    mX = x;  
}  
  
float getYLoc() {  
    return mY;  
}  
  
void setYLoc(float y) {  
    mY = y;  
}  
  
void setTouches(int touches) {  
    mTouches = touches;  
}  
  
@Override  
protected void onDraw(Canvas canvas) {  
    canvas.drawCircle(mX, mY, MAX_SIZE / mTouches, mPaint);  
}  
}
```

# GESTUREDETECTOR

A CLASS THAT RECOGNIZES COMMON  
TOUCH GESTURES

SOME BUILT-IN GESTURES INCLUDE  
CONFIRMED SINGLE TAP, DOUBLE TAP,  
FLING

# GESTUREDETECTOR

ACTIVITY CREATES A GestureDetector  
THAT IMPLEMENTS THE GestureDetector.  
OnGestureListener interface

ACTIVITY WILL RECEIVE CALLS TO  
onTouchEvent() WHEN ACTIVITY IS  
TOUCHED

ONTouchEvent DELEGATES CALL TO  
GestureDetector.OnGestureListener

# TOUCHGESTUREVIEWFLIPPER

SHOWS A TextView DISPLAYING A NUMBER

IF THE USER PERFORMS A RIGHT TO LEFT  
“FLING” GESTURE,

THE TextView WILL SCROLL OFF THE SCREEN

A NEW TextView WILL SCROLL IN BEHIND IT



# TOUCHGESTUREVIEWFLIPPER

```
public class ViewFlipperTestActivity extends Activity {  
    private ViewFlipper mFlipper;  
    private TextView mTextView1, mTextView2;  
    private int mCurrentLayoutState, mCount;  
    private GestureDetector mGestureDetector;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        mCurrentLayoutState = 0;  
    }  
}
```



# TOUCHGESTUREVIEWFLIPPER

```
mFlipper = (ViewFlipper) findViewById(R.id.view_flipper);
mTextView1 = (TextView) findViewById(R.id.textView1);
mTextView2 = (TextView) findViewById(R.id.textView2);

mTextView1.setText(String.valueOf(mCount));

mGestureDetector = new GestureDetector(this,
    new GestureDetector.SimpleOnGestureListener() {
        @Override
        public boolean onFling(MotionEvent e1, MotionEvent e2,
            float velocityX, float velocityY) {
            if (velocityX < -10.0f) {
                mCurrentLayoutState = mCurrentLayoutState == 0 ? 1
                    : 0;
                switchLayoutStateTo(mCurrentLayoutState);
            }
            return true;
        }
    });
}
```

# TOUCHGESTUREVIEWFLIPPER

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

public void switchLayoutStateTo(int switchTo) {
    mCurrentLayoutState = switchTo;

    mFlipper.setInAnimation(inFromRightAnimation());
    mFlipper.setOutAnimation(outToLeftAnimation());

    mCount++;

    if (switchTo == 0) {
        mTextView1.setText(String.valueOf(mCount));
    } else {
        mTextView2.setText(String.valueOf(mCount));
    }

    mFlipper.showPrevious();
}
```

# TOUCHGESTUREVIEWFLIPPER

```
private Animation inFromRightAnimation() {
    Animation inFromRight = new TranslateAnimation(
        Animation.RELATIVE_TO_PARENT, +1.0f,
        Animation.RELATIVE_TO_PARENT, 0.0f,
        Animation.RELATIVE_TO_PARENT, 0.0f,
        Animation.RELATIVE_TO_PARENT, 0.0f);
    inFromRight.setDuration(500);
    inFromRight.setInterpolator(new LinearInterpolator());
    return inFromRight;
}

private Animation outToLeftAnimation() {
    Animation outToLeft = new TranslateAnimation(
        Animation.RELATIVE_TO_PARENT, 0.0f,
        Animation.RELATIVE_TO_PARENT, -1.0f,
        Animation.RELATIVE_TO_PARENT, 0.0f,
        Animation.RELATIVE_TO_PARENT, 0.0f);
    outToLeft.setDuration(500);
    outToLeft.setInterpolator(new LinearInterpolator());
    return outToLeft;
}
```

# CREATING CUSTOM GESTURES

THE GESTUREBUILDER APPLICATION LETS YOU  
CREATE & SAVE CUSTOM GESTURES

COMES BUNDLED WITH SDK

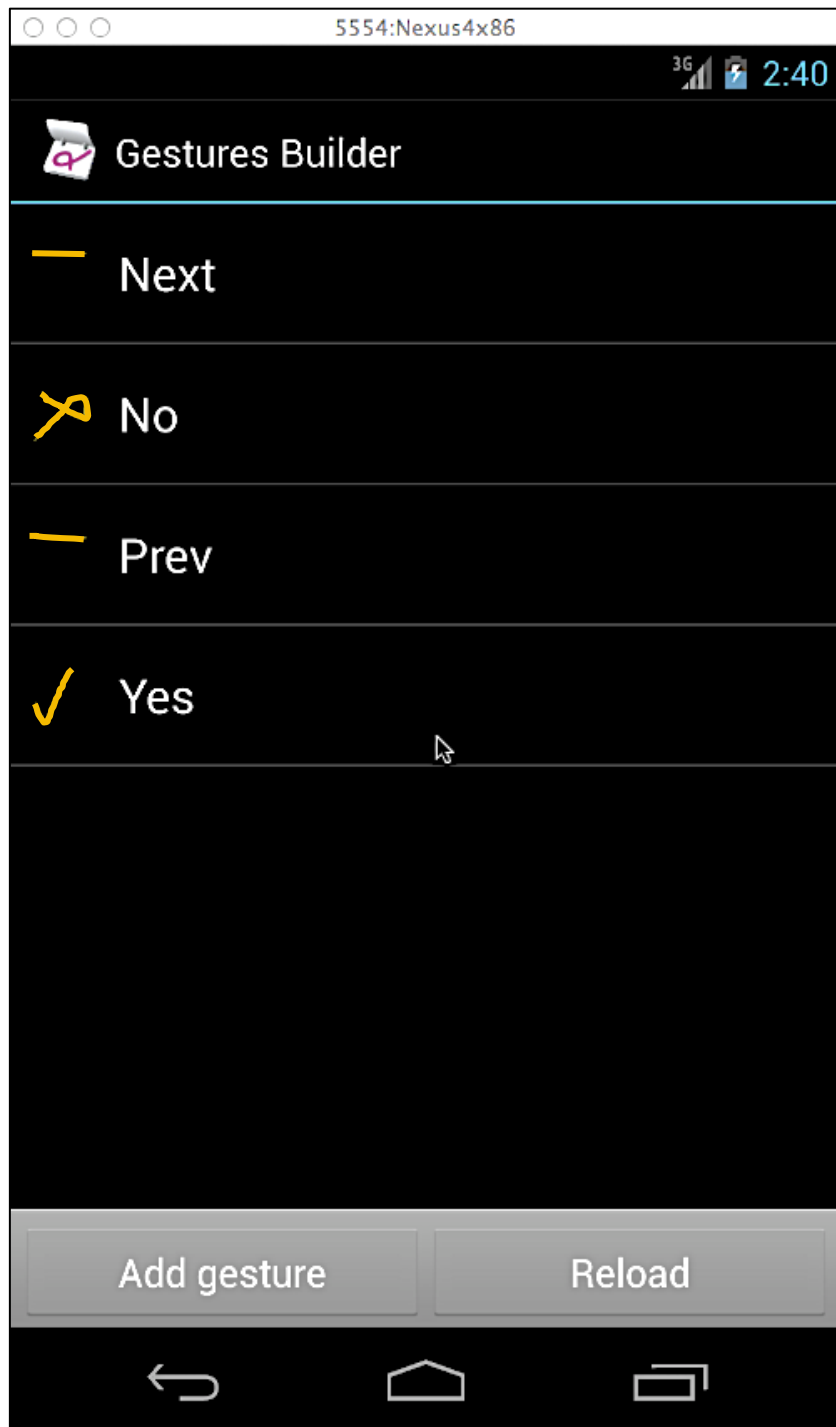
# CREATING CUSTOM GESTURES

GestureLibraries SUPPORTS LOADING  
CUSTOM GESTURES & THEN RECOGNIZING  
THEM AT RUNTIME

# CREATING CUSTOM GESTURES

INCLUDE A `GestureOverlayView` IN YOUR LAYOUT

THE OVERLAY INTERCEPTS USER GESTURES AND INVOKES YOUR APPLICATION CODE TO HANDLE THEM



# GESTUREBUILDER

# GESTUREBUILDER

STORES GESTURES TO  
/mnt/sdcard/gestures

COPY THIS FILE TO  
/res/raw DIRECTORY

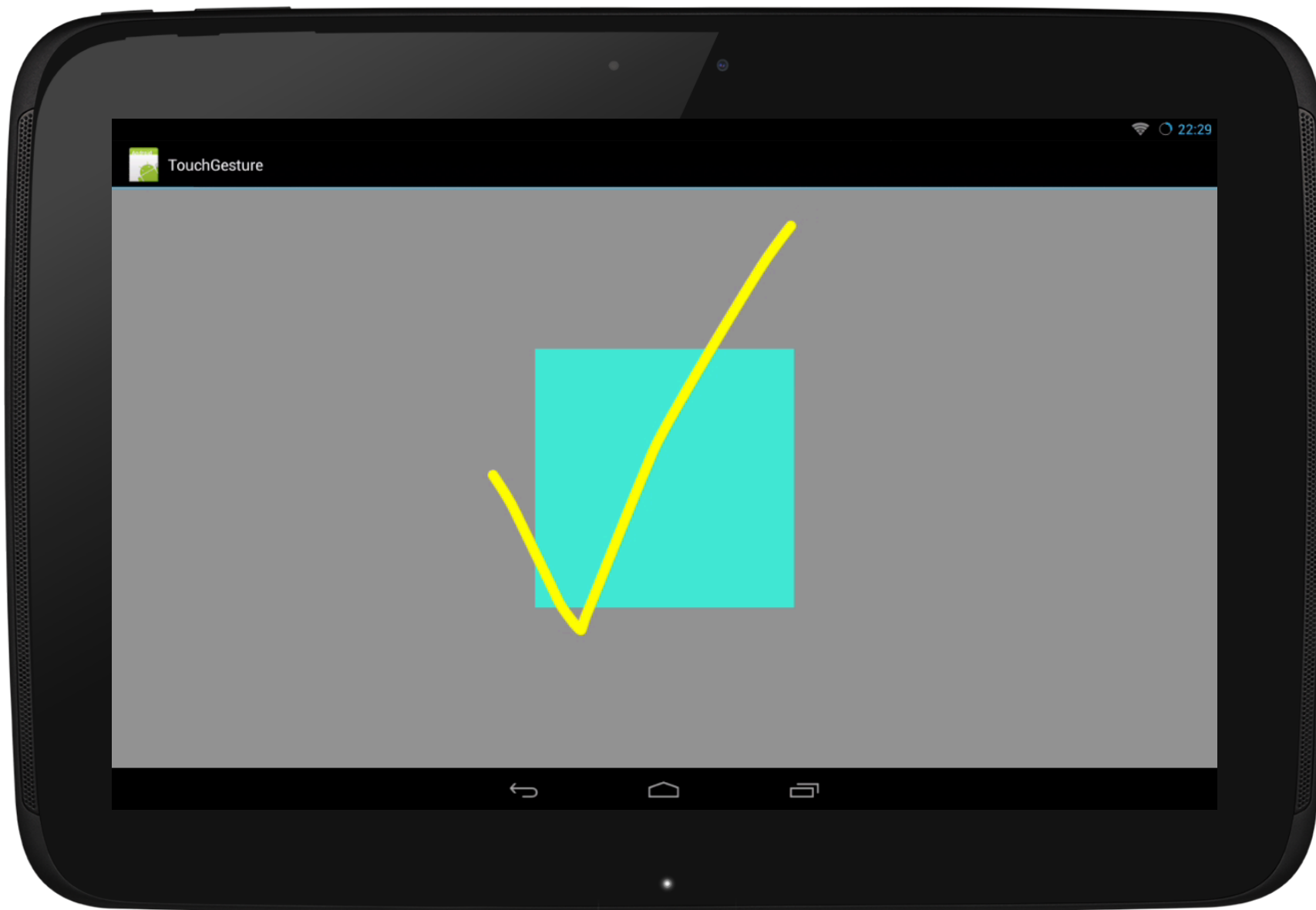


# TOUCHGESTURES

APPLICATION DISPLAYS A SMALL View WITH  
A COLORED BACKGROUND

USER CAN SWIPE LEFT AND RIGHT TO CYCLE  
BETWEEN DIFFERENT CANDIDATE  
BACKGROUND COLORS

CAN MAKE AN CHECK OR X-LIKE GESTURE  
TO SET OR CANCEL THE APPLICATION'S  
CURRENT BACKGROUND COLOR



# TOUCHGESTURES

```
public class GesturesActivity extends Activity implements
    OnGesturePerformedListener {
    private static final String NO = "No";
    private static final String YES = "Yes";
    private static final String PREV = "Prev";
    private static final String NEXT = "Next";
    private GestureLibrary mLibrary;
    private int mBgColor = 0;
    private int mFirstColor, mStartBgColor = Color.GRAY;
    private FrameLayout mFrame;
    private RelativeLayout mLayout;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

# TOUCHGESTURES

```
mFrame = (FrameLayout) findViewById(R.id.frame);
mBgColor = new Random().nextInt(0xFFFFFFFF) | 0xFF000000;
mFirstColor = mBgColor;
mFrame.setBackgroundColor(mBgColor);

mLayout = (RelativeLayout) findViewById(R.id.main);
mLayout.setBackgroundColor(mStartBgColor);

mLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
if (!mLibrary.load()) {
    finish();
}

// Make this the target of gesture detection callbacks
GestureOverlayView gestureView = (GestureOverlayView) findViewById(R.id.gestures_overlay);
gestureView.addOnGesturePerformedListener(this);
}
```

# TOUCHGESTURES

```
public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture) {  
  
    // Get gesture predictions  
    ArrayList<Prediction> predictions = mLibrary.recognize(gesture);  
  
    // Get highest-ranked prediction  
    if (predictions.size() > 0) {  
        Prediction prediction = predictions.get(0);  
  
        // Ignore weak predictions  
  
        if (prediction.score > 2.0) {  
            if (prediction.name.equals(PREV)) {  
  
                mBgColor -= 100;  
                mFrame.setBackgroundColor(mBgColor);  
  
            } else if (prediction.name.equals(NEXT)) {  
  
                mBgColor += 100;  
                mFrame.setBackgroundColor(mBgColor);  
  
            }  
        }  
    }  
}
```

# TOUCHGESTURES

```
    } else if (prediction.name.equals(YES)) {  
        mLayout.setBackgroundColor(mBgColor);  
    } else if (prediction.name.equals(NO)) {  
        mLayout.setBackgroundColor(mStartBgColor);  
        mFrame.setBackgroundColor(mFirstColor);  
    } else {  
        Toast.makeText(this, prediction.name, Toast.LENGTH_SHORT)  
            .show();  
    }  
} else {  
    Toast.makeText(this, "No prediction", Toast.LENGTH_SHORT)  
        .show();  
}  
}  
}
```

NEXT TIME

MULTIMEDIA