CSC 472 / 372 Mobile Application Development for Android

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Outline

- Handling touch input & events
- *Motion Event* and listeners
- Handling multi-touch
- Gestures
- *Gesture Detectors*
- Multi-touch gestures

Touch Input & Events

Touch Input & Events

- When a user places one or more fingers on the touch screen.
- The movement of the fingers are captured through a series of *Motion Events*
- Each *Motion Event* contains
  - an *action code*, e.g., finger up/down, and
  - a set of *axis values*, e.g., position, pressure, etc.
- System dispatches *Motion Events* to the *View* object that is being touched, known as the *target*.

Not to be confused with *Sensor Events*

The details of event dispatching later

Receive Touch Events

- *Views* and *Activities* can receive and handle *Motion Events*
- Several ways to receive touch events
  a) Extend a *View* class, i.e., a custom *View*, and override the `onTouchEvent()` callback method.
  b) Override the `onTouchEvent()` callback method in an *Activity*
     - Assuming none of the *View* objects in its content view hierarchy is handling touch events.
  c) Set an *On Touch Listener* to a *View*.
     - Handling touch events originating from that *View*

Motion Event Class Methods

- Get the action code
  `int getAction()`
  - `ACTION_DOWN` A finger touched down, i.e., the start of a gesture
  - `ACTION_MOVE` A finger moved, i.e., during a gesture
  - `ACTION_UP` A finger lifted up, i.e., the end of a gesture
- Get the location, the X-Y coordinate, of the touch
  `float getX()`
  `float getY()`

Batching Motion Event

- The `ACTION_MOVE` events may batch together multiple movement samples within a single event.
- The most recent, or current, location returned as: `getX()` and `getY()`.
- Additional locations are reported as a history of locations
  - Older than the current location in the batch
  - More recent than other locations reported in prior events
- Accessed the history
  - `getHistorySize()`
  - `getHistoricalX(int pos)` `getHistoricalY(int pos)`

The onTouchEvent() Method

```
@Override public boolean onTouchEvent(MotionEvent event) {
  int action = event.getAction();
  switch (action) {
    case MotionEvent.ACTION_DOWN:
      … process event …
      return true;
    case MotionEvent.ACTION_MOVE:
      … process event …
      return true;
    case MotionEvent.ACTION_UP:
      … process event …
      return true;
    default :
      return super.onTouchEvent(event);
  }
}
```

Return `true`: the event has been consumed.
Return `false`: otherwise

Return false implies
- You are not interested
- Future events in the same gesture will not be dispatched to this object.

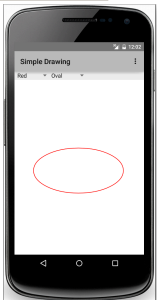
The OnTouchListener

```
View myView = findViewById(R.id.my_view);
myView.setOnTouchListener(new OnTouchListener() {
  public boolean onTouch(View v, MotionEvent event) {
    int action = event.getAction();
    switch (action) {
      case MotionEvent.ACTION_DOWN:  … process event …
        return true;
      case MotionEvent.ACTION_MOVE:  … process event …
        return true;
      case MotionEvent.ACTION_UP:    … process event …
        return true;
    }
    return false;
  }
});
```

Similar processing as in `onTouchEvent()`

A Simple Drawing App

- Draw simple shapes using single touch.
  - Trace the finger movement
  - Draw
    - Line
    - Rectangle
    - Oval
- Choose color and shape
- Handle basic touch events



The Drawing App – A Custom View

```
public class MyView extends View {
  enum ShapeType {Trace, Line, Rectangle, Oval};

  private int color = Color.BLUE;
  private ShapeType shapeType = ShapeType.Line.Trace;

  public MyView(Context context) { … }
  public MyView(Context context, AttributeSet attrs) { … }

  public void setColor(String name) { … }
  public void setShapeType(String name) { … }

  @Override protected void onDraw(Canvas canvas) { … }
  @Override public boolean onTouchEvent(MotionEvent event) { … }
}
```

Values set by the spinners in UI

Used by the *Activity* to set the values.

Next several slides.

The Custom View – Handle Touch Events, 1/3

```
enum ShapeType {Trace, Line, Rectangle, Oval};
private Path path = new Path();
private ShapeType shapeType = ShapeType.Trace;
float startX, startY;
public boolean onTouchEvent(MotionEvent event) {
  int action = event.getAction();
  float curX = event.getX();
  float curY = event.getY();
  switch (action) {
    case MotionEvent.ACTION_DOWN:
      startX = curX;
      startY = curY;
      if (shapeType == ShapeType.Trace) {
        path.reset();
        path.moveTo(curX, curY);
      }
      break;
  …
```

Building a path.

The current action code and position.

The start position of a shape.

The Custom View – Handle Touch Events, 2/3

```
switch (action) {
  case MotionEvent.ACTION_DOWN: …
  case MotionEvent.ACTION_MOVE:
  case MotionEvent.ACTION_UP:
    if (shapeType == ShapeType.Trace) {
      final int historySize = event.getHistorySize();
      for (int h = 0; h < historySize; h++) {
        float histX = event.getHistoricalX(h);
        float histY = event.getHistoricalY(h);
        path.lineTo(histX, histY);
      }
      path.lineTo(curX, curY);
    } else  if (shapeType == ShapeType.Line) {
      path.reset();
      path.moveTo(startX, startY);
      path.lineTo(curX, curY);
    } else …
```

ACTION_MOVE and ACTION_UP are handled in the same way.

Get historical locations.

A straight line.

The Custom View – Handle Touch Events, 3/3

```
switch (action) {
  case MotionEvent.ACTION_DOWN: …
  case MotionEvent.ACTION_MOVE:
  case MotionEvent.ACTION_UP:
    if (shapeType == ShapeType.Trace) { …
    } else  if (shapeType == ShapeType.Line) { …
    } else  if (shapeType == ShapeType.Rectangle) {
      path.reset();
      path.addRect(startX, startY, curX, curY, Path.Direction.CW);
    } else  if (shapeType == ShapeType.Oval) {
      path.reset();
      path.addOval(startX, startY, curX, curY, Path.Direction.CW);
    }
    break;
}
invalidate();
return true;
}
```

A rectangle.

An oval.

The path is drawn each time a *Motion Event* is received.

The Custom View – Handle Drawing

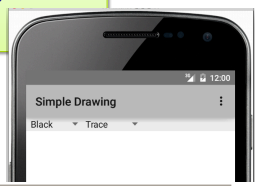
```
private Path path = new Path();
private Paint paint = new Paint();
private int color = Color.BLUE;
@Override
protected void onDraw(Canvas canvas) {
  canvas.drawColor(Color.WHITE);

  paint.setAntiAlias(true);
  paint.setColor(color);
  paint.setStyle(Paint.Style.STROKE);
  paint.setStrokeJoin(Paint.Join.ROUND);
  paint.setStrokeWidth(3);

  canvas.drawPath(path, paint);
}
```

Draw the path.

DePaul University 23

The Drawing App – The Layout

```
<LinearLayout android:orientation="vertical" … >
  <LinearLayout … >
    <Spinner android:id="@+id/colors" … />
    <Spinner android:id="@+id/shapes" … />
  </LinearLayout>
  <edu.depaul.csc472.simpledrawing.MyView
    android:id="@+id/v1" … />
</LinearLayout>
```



DePaul University 24

The Drawing App – The Activity

```
public class MainActivity extends Activity {
  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final MyView v1 = (MyView) findViewById(R.id.v1);
    final Spinner s1 = (Spinner) findViewById(R.id.colors); …
    s1.setOnItemSelectedListener(
      new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
                                   View view, int position, long id) {
          v1.setColor(COLORS[position]);
        }
        public void onNothingSelected(AdapterView<?> parent) { }
    });
  …
}
```

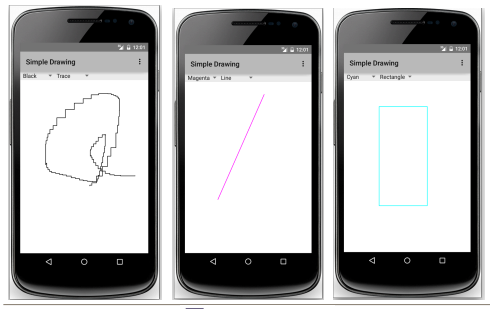
DePaul University 25

The Drawing App – The Activity

```
public class MainActivity extends Activity {
  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final MyView v1 = (MyView) findViewById(R.id.v1);
    …
    final Spinner s2 = (Spinner) findViewById(R.id.shapes); …
    s2.setOnItemSelectedListener(
      new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
                                   View view, int position, long id) {
          v1.setShapeType(SHAPES[position]);
        }
        public void onNothingSelected(AdapterView<?> parent) { }
    });
  }
```

DePaul University 26

The Drawing App



Motion Events Dispatching

- *Motion Events* are dispatched top-down from the root container of the view hierarchy to the widget that is first touched, the target.
  - A *View Group* may intercept the events, and its children will not receive the events
- *Motion Events* in the same continuous touch sequence are continue to be dispatched to the target, the widget first touched
  - even if the touch locations have moved outside the region occupied by the target.

Motion Event Handling

- Motion Events are handled bottom-up in the view hierarchy
  - The target *View* object touched has the first opportunity
  - If a *View* object does not consume in the event, the event will be passed to its parent *View Group*.
  - A *View* object may declare itself uninterested, and the subsequent events in the same touch sequence will no longer be dispatched to it.
- The active *Activity* will receive the events if they are not consumed by the content view hierarchy.

Handling Multi-Touch

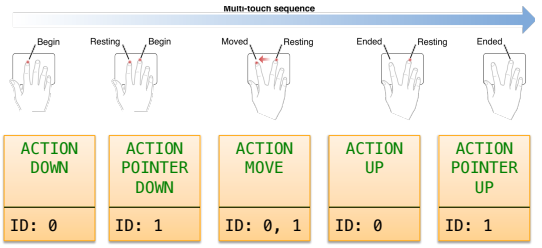
Motion Events for Multi-Touch

- For multi-touch screens, *Motion Events* are generated for each touch source, i.e., finger
- Individual fingers are referred to as *pointers*
- Each pointer has a unique ID
  - Assigned when it first touches down
  - Stay the same during a continuous touch sequence, until it lifts up
- *Motion Events* contain information about all of the pointers that are currently active, i.e., in contact with the screen

Multi-Touch Event Sequence

- Separate event for each pointer touches down
  - `ACTION_DOWN` for the first, i.e., primary, pointer
  - `ACTION_POINTER_DOWN` for the extra pointers
- Pointers move as a group
  - An `ACTION_MOVE` event may involve movement of several pointers
- Separate event for each pointer lifts up
  - `ACTION_POINTER_UP` for the extra pointers
  - `ACTION_UP` for the primary pointer

Multi-Touch Event Sequence



Motion Event Class Methods

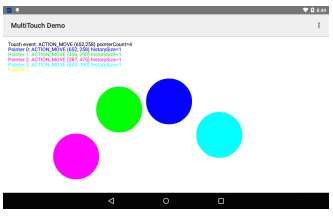
- Action for individual pointers
  `getActionMasked()`
  - `ACTION_POINTER_DOWN`
  - `ACTION_POINTER_UP`
- The index of a pointer
  `getActionIndex()`
  - The array index at which the data for this pointer is stored
  - Most methods use the index as a parameter, not Id
- The Id of a pointer
  `getPointerId(int i)`
  - The Id of the pointer at index *i*

Motion Event Class Methods

- The number of active pointers in this event
  `getPointerCount()`
- Access pointer locations
  `getX(int i)`
  `getY(int i)`
  `getHistoricalX(int i, int pos)`
  `getHistoricalY(int i, int pos)`
  - *i* is the index of the pointer
  - *pos* is the position in history

Multi-Touch Demo

- A demo app handles multi-touch
- Access data on all pointers
- Access history
- Display event data
- Draw touch points visually
- Log all events



Multi-Touch Demo App – The Custom View

```
public class MyView extends View {

  public MyView(Context context) { … }
  public MyView(Context context, AttributeSet attrs) { … }

  @Override
  public boolean onTouchEvent(MotionEvent event) { … }

  void logEvent(MotionEvent event) { … }

  @Override
  protected void onDraw(Canvas canvas) { … }
}
```


The Custom View – Handle Touch Events

```
public boolean onTouchEvent(MotionEvent event) {
  logEvent(event);
  invalidate();
  …
  return true;
}
```

1. Retrieve the data of the event.
2. Request drawing.

Will elaborate later

The Custom View – Multi-Touch Event Data

```
String eventMessage = "No Event";

int pointerCount;

String[] pointers = new String[5];

List<Point>[] locations = new List[5];

void logEvent(MotionEvent event) { … }
```

A message with action code and location

# of pointers

A message for each active pointer, up to 5

A list of locations for each active pointer, including historical locations

Collect data for each event

DePaul University 46

The Custom View – Collect Event Data, 1/2

```
void logEvent(MotionEvent event) {
  for (int i = 0; i < 5; i++) {
    pointers[i] = null;
    locations[i] = null;
  }
  int action = event.getActionMasked();
  float eventX = event.getX();
  float eventY = event.getY();
  int historySize = event.getHistorySize();
  pointerCount = event.getPointerCount();
  String actionName = MotionEvent.actionToString(action);
  if (action == MotionEvent.ACTION_POINTER_DOWN ||
      action == MotionEvent.ACTION_POINTER_UP) {
    actionName += (" " + event.getActionIndex() );
  }
  …
```

Clear the data from previous event

DePaul University 48

The Custom View – Collect Event Data, 2/2

```
void logEvent(MotionEvent event) { …
  eventMessage = actionName + … ;
  for (int p = 0; p < pointerCount; p++) {
    int pointerId = event.getPointerId(p);
    int pointerX = (int) event.getX(p);
    int pointerY = (int) event.getY(p);
    if (pointerId >= 0 && pointerId < 5) {
      pointers[pointerId] = actionName + …;
      locations[pointerId] = new ArrayList<Point>();
      for (int h = 0; h < historySize; h++) {
        int pointerHX = (int) event.getHistoricalX(p, h);
        int pointerHY = (int) event.getHistoricalY(p, h);
        locations[pointerId].add(new Point(pointerHX, pointerHY));
      }
      locations[pointerId].add(new Point(pointerX, pointerY));
    }
  }
}
```

Location for each pointer

Historical locations for each pointer

The Custom View – Displaying the Event Data

```
@Override protected void onDraw(Canvas canvas) {
  canvas.drawColor(Color.WHITE);
  paint.setAntiAlias(true); …
  canvas.drawText("Touch event: " + eventMessage, 20, 40, paint);
  for (int p = 0; p < 5; p++) {
    paint.setColor(colors[p]);
    String msg = "Pointer " + p + ": ";
    if (pointers[p] != null) {
      int size = locations[p].size();
      for (Point point : locations[p]) {
        canvas.drawCircle(point.x, point.y,
                          50 + 10 * pointerCount, paint);
      }
      msg += pointers[p];
    }
    canvas.drawText(msg, 20, 60 + 20 * p, paint);
  }
}
```

Draw circles for each pointer

The Custom View – Handle Touch Events, Revisited

```
public boolean onTouchEvent(MotionEvent event) {
  logEvent(event);
  invalidate();
  int action = event.getAction();
  if (action == MotionEvent.ACTION_UP) {
    handler.postDelayed(new Runnable() {
      @Override public void run() {
        eventMessage = "No Event";
        for (int i = 0; i < 5; i++) {
          pointers[i] = null; locations[i] = null;
        }
        invalidate();
      }
    }, 1000);
  }
  return true;
}
```

Handling Gestures

Gestures

- A sequence of touches and/or movements with one or more fingers on the touch screen
- Particular patterns of touches and/or movements are interpreted by apps as gestures.
- Common gestures
  - Touch, tap (single tap), double tap
  - Long press
  - Scroll (drag), fling (swipe)
  - Scale (pinch)

Gesture Detector

- A class that detects common gestures, a.k.a. *gesture recognizer*
- Works in conjunction with the `onTouchEvent()` method
  - Must forward all events to the *Gesture Detector*
- Requires a listener, which listens to the callbacks from the *Gesture Detector*
  - A listener that implements the callback interface `GestureDetector.OnGestureListener`, or
  - A listener that extends the class `GestureDetector.SimpleOnGestureListener`

Gesture Listeners

- Interface `GestureDetector.OnGestureListener` provides the following callbacks
  - `onDown()` a finger touches down
  - `onSingleTapUp()` a finger lifts up
  - `onShowPress()` press and hold
  - `onLongPress()` press, hold and lift
  - `onScroll()` press and move
  - `onFling()` touch, quick move and lift
- Multiple notifications may occur during a single gesture
  - Some notifications are precursors to others

Handle Single & Double Taps

- Single tap is a precursor to a double tap
- If you wish to differentiate single and double taps, implement the `GestureDetector.OnDoubleTapListener` interface
- It provides the following callbacks
  - `onSingleTapConfirmed()` A definitive single tap
  - `onDoubleTapEvent()` A precursor to a double tap
  - `onDoubleTap()` A definitive double tap

Support Backward Compatibility

- Touch screen technology, and the design and implementation of *Gesture Detector* related API have evolved over the years
  - Not all versions of Android have the same capability and implementation
- Android provides a number of *support libraries* to maximize the backward compatibility in several areas
- The *Gesture Detector* implementation is in the main library
- An alternative implementation *Gesture Detector Compat* is in the support library

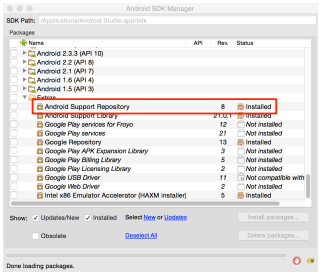
Android Support Libraries

- APIs for additional features, and backward-compatibility
  - Improve looks, increase performance, and broaden reach
  - Support newer API in older versions
  - **Best practice to include support libraries**
- Each library targets a base Android API level and provides a different set of features.
  - v4 – compatible with API-level 4 (Android 1.6) or higher
  - v7 – compatible with API-level 7 (Android 2.1) or higher
  - Also, v8, v13, and v17

Add Android Support Libraries to Android Studio

1. In *SDK Manager*

   Make sure the *Android Support Repository* is downloaded



Add Android Support Libraries to Android Studio

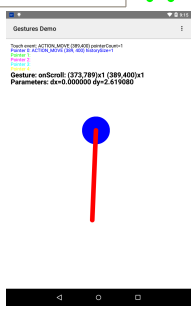
2. Edit the **build.gradle** (Module: app) file of your application.
   - Add the support library to the dependencies section.
   - e.g., to add the v4 support library, add the following lines (for build tools version 23):

```
dependencies {
  ...
  compile "com.android.support:support-v4:23.0.1"
}
```

Build tool version. Needs to be updated when SDK tools are updated

Gestures Demo App

- Detect all the common gestures supported by the *Gesture Detector*
  - Single tap
  - Double tap
  - Long press
  - Scroll/drag
  - Fling
- An extension of the *Multi-Touch Demo* app



The Gesture Demo App – The Custom View

```
public class MyView extends View
implements GestureDetector.OnGestureListener,
           GestureDetector.OnDoubleTapListener {

  private GestureDetectorCompat gestureDetector;

  public MyView(Context context, AttributeSet attrs) {
    super(context, attrs);
    gestureDetector =
        new GestureDetectorCompat(context, this);
    gestureDetector.setOnDoubleTapListener(this);
  }
  …
```

Use the compatibility class

The listener

The Custom View – Handle Touch Events

```
public boolean onTouchEvent(MotionEvent event) {
  gestureDetector.onTouchEvent(event);
  logEvent(event);
  invalidate();
  int action = event.getAction();
  if (action == MotionEvent.ACTION_UP) {
    getHandler().postDelayed(new Runnable() {
      @Override public void run() {
        eventMessage = "No Event";
        gestureMessage = "";  gestureParam = "";
        start = end = null; …
        invalidate();
      }
    }, 1000);
  }
  return true;
}
```

Forward events to the *Gesture Detector*

The Custom View – Collecting Gesture Data

```
String gestureMessage = "";
String gestureParam = "";
Point start, end;
private void logGesture(String msg, String param,
                        MotionEvent e1, MotionEvent e2) {
  Log.d(TAG, msg + ": e1=" + e1 + " e2=" + e2);
  int pointerCount1 = e1.getPointerCount();
  int pointerCount2 = e2.getPointerCount();
  gestureMessage = msg + String.format(": (%d,%d)x%d (%d,%d)x%d",
        (int) e1.getX(), (int) e1.getY(), pointerCount1,
        (int) e2.getX(), (int) e2.getY(), pointerCount2);
  gestureParam = param;
  start = new Point((int) e1.getX(), (int) e1.getY());
  end = new Point((int) e2.getX(), (int) e2.getY());
}
```

The message and parameters for the detected gesture. The start and end points.

The Double Tap Listener Callbacks – Single & Double Taps

```
@Override public boolean onSingleTapConfirmed(MotionEvent e) {
  logGesture("onSingleTapConfirmed", e);
  return true;
}
@Override public boolean onDoubleTapEvent(MotionEvent e) {
  logGesture("onDoubleTapEvent", e);
  return true;
}
@Override public boolean onDoubleTap(MotionEvent e) {
  logGesture("onDoubleTap", e);
  return true;
}
```

Definitive single tap

Definitive double tap

Return true: the event is handled.
Return false: otherwise

DePaul University 76

The Gesture Listener Callbacks – Tap and Long Press

```
@Override public boolean onDown(MotionEvent e) {
  logGesture("onDown", e);
  return true;
}

@Override public boolean onSingleTapUp(MotionEvent e) {
  logGesture("onSingleTapUp", e);
  return true;
}
@Override public void onShowPress(MotionEvent e) {
  logGesture("onShowPress", e);
}
@Override public void onLongPress(MotionEvent e) {
  logGesture("onLongPress", e);
}
```

Must return true to receive further callbacks in this gesture.

DePaul University 78

The Gesture Listener Callbacks – Scroll & Fling

```
@Override public boolean onScroll(MotionEvent e1, MotionEvent e2,
                        float distanceX, float distanceY) {
  logGesture("onScroll", String.format("dx=%f dy=%f",
            distanceX, distanceY), e1, e2);
  return true;
}

@Override public boolean onFling(MotionEvent e1, MotionEvent e2,
                        float velocityX, float velocityY) {
  logGesture("onFling", String.format("vx=%f vy=%f",
            velocityX, velocityY), e1, e2);
  return true;
}
```

DePaul University 79

The Custom View – Displaying the Gesture Data

```
@Override protected void onDraw(Canvas canvas) {
     canvas.drawColor(Color.WHITE);
  …

  paint.setColor(Color.BLACK);
  paint.setTypeface(Typeface.DEFAULT_BOLD);
  paint.setTextSize(30);
  canvas.drawText("Gesture: " + gestureMessage, 20, 170, paint);
  canvas.drawText("Parameters: " + gestureParam, 20, 200, paint);
  if (start != null && end != null) {
    paint.setColor(Color.RED);
    paint.setStrokeWidth(20);
    paint.setStrokeCap(Paint.Cap.ROUND);
    canvas.drawLine(start.x, start.y, end.x, end.y, paint);
  }
}
```

Draw a line from the start to the end point of the gesture.

ERSITY 81

Support a Subset of Gestures

- If you only want to use a subset of the gestures recognized by the *Gesture Detector*, you can extend the `GestureDetector.SimpleOnGestureListener` class
- Provide default implementations for all callback methods
- Only override the callback methods you are interested in.

DePaul University 82

Simple Gesture Listener

```
public class MainActivity extends Activity {
  private GestureDetectorCompat mDetector;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mDetector = new GestureDetectorCompat(this,
                    new MyGestureListener());
  }
  @Override
  public boolean onTouchEvent(MotionEvent event){
    this.mDetector.onTouchEvent(event);
    return super.onTouchEvent(event);
  }
  …
```

DePaul University 83

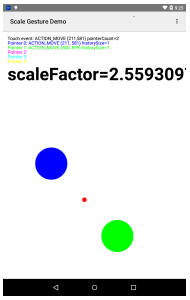
Simple Gesture Listener

```
public class MainActivity extends Activity {
  private GestureDetectorCompat mDetector;
  …
  class MyGestureListener
        extends GestureDetector.SimpleOnGestureListener {
    @Override public boolean onDown(MotionEvent event) {
      return true;
    }

    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
                           float velocityX, float velocityY) {
      … process fling gesture …
      return true;
    }
  }
}
```

Must return true to receive further callbacks in this gesture. Default is false.

Scale Gesture Demo

- Recognize the scale gesture
- Use the scale factor of the gesture to scale the text size of the message
- An extension of the *Multi-Touch Demo* app



Scale Gesture Detector

- Recognizes the scale gesture, a.k.a. pinch
  - Two fingers press and move inward or outward
- Uses a callback interface `ScaleGestureDetector.OnScaleGestureListener`.
  - Callback methods defined
    `onScale()`
    `onScaleBegin()`
    `onScaleEnd()`
- Also provides a helper class `ScaleGestureDetector.SimpleOnScaleGestureListener`

Scale Gesture App

```
public class MyView extends View {
  private ScaleGestureDetector gestureDetector;
  private float scaleFactor = 1.f;
  public MyView(Context context, AttributeSet attrs) {
    super(context, attrs);
    gestureDetector =
        new ScaleGestureDetector(context, new ScaleListener());
  }
  @Override protected void onDraw(Canvas canvas) { … }
  @Override public boolean onTouchEvent(MotionEvent event) { … }
  void logEvent(MotionEvent event) {

  private class ScaleListener
    extends ScaleGestureDetector.SimpleOnScaleGestureListener { … }
}
```

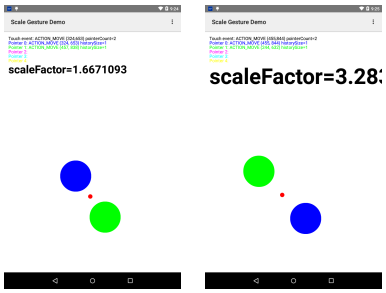

Scale Gesture App – The Scale Listener

```
Point focus;

private class ScaleListener
  extends ScaleGestureDetector.SimpleOnScaleGestureListener {
  @Override
  public boolean onScale(ScaleGestureDetector detector) {
    scaleFactor *= detector.getScaleFactor();
    scaleFactor = Math.max(0.1f, Math.min(scaleFactor, 5.0f));
    gestureMessage = "scaleFactor=" + scaleFactor;
    focus = new Point((int) detector.getFocusX(),
                      (int) detector.getFocusY());
    return true;
  }
}
```


Scale Gesture App – Drawing

```
@Override protected void onDraw(Canvas canvas) {
  …
  int size = (int) (30 * scaleFactor);
  paint.setTextSize(size);
  paint.setColor(Color.BLACK);
  paint.setTypeface(Typeface.DEFAULT_BOLD);
  canvas.drawText(gestureMessage, 20, 140 + size, paint);

  if (focus != null) {
    paint.setColor(Color.RED);
    paint.setStrokeWidth(20);
    paint.setStrokeCap(Paint.Cap.ROUND);
    canvas.drawCircle(focus.x, focus.y, 10, paint);
  }
}
```

Scale Gesture App



The Sample Code

- The sample apps in this lecture are available in D2L
  - SimpleDrawing.zip
  - MultiTouchDemo.zip
  - GestureDemo.zip
  - ScaleGestureDemo.zip
- Each zip archive contains the entire project folder
- Unzip the file and import to Android Studio

Next …

- Sensor overview
- Motion sensors
- Accelerometer
- Gyroscope