ATLS 4120/5120: Mobile Application Development
Week 15: Gestures

Without a keyboard or mouse users interact with mobile devices mainly through touch and the use of gestures. As a developer you can take advantage of these touch events, just make sure your apps always conform to mobile app accepted conventions so they're easy for users to figure out and use.

Android Gestures https://material.io/guidelines/patterns/gestures.html#

Touch mechanics refer to what the user's fingers do on the screen.
A touch activity may be achieved through combining multiple touch mechanics.
Gestures include touch mechanics and touch activities.
Gesture are basically a series or pattern of touch events that you can decide how you want to process.

When a user places one or more fingers on the screen, this triggers the callback onTouchEvent() on the view that received the touch events. For each sequence of touch events (position, pressure, size, addition of another finger, etc.) that is identified as a gesture, onTouchEvent() is fired several times.
The gesture starts when the user first touches the screen, continues as the system tracks the position of the user's finger(s), and ends by capturing the final event of the user's fingers leaving the screen. Throughout this interaction, the MotionEvent delivered to onTouchEvent() provides the details of every interaction.

Detect Common Gestures
https://developer.android.com/training/gestures/detector.html
If your app uses common gestures such as double tap, long press, fling, etc, you can take advantage of the GestureDetector class. The drag motion is when a user is moving something across the screen. Scrolling is the general process of moving the view while pan is when the scrolling motion causes scrolling in both the x and y axes.
GestureDetector makes it easy for you to detect common gestures without processing the individual touch events yourself. Implementing the GestureDetector.OnGestureListener and GestureDetector.OnDoubleTapListener interfaces and overriding Activity's onTouchEvent(MotionEvent event) method lets all the events be passed to your activity. There you implement the required methods from the interfaces to process the gestures. A return value of true from the individual on<TouchEvent> methods indicates that you have handled the touch event. A return value of false passes events down through the view stack until the touch has been successfully handled.

If you only want to process a few gestures, you can implement the GestureDetector.SimpleOnGestureListener interface and then only override the methods you're interested in (you can omit the others).
You must ALWAYS implement an onDown() method that returns true. This is because all gestures begin with an onDown() message. If you return false from onDown(), as GestureDetector.SimpleOnGestureListener does by default, the system assumes that you want to ignore the rest of the gesture, and the other methods of GestureDetector.OnGestureListener never get called.

Depending on what you're trying to achieve you can track different types of motion:

- Velocity
- Start and end points of a touch
- Direction
- History

MotionEvents touches are accessible in the onTouch(View v, MotionEvent event) method.

- Each finger that touches the screen is referred to as a pointer
- Each pointer is stored in an array
    - Index: position of the pointer in the array
    - ID: each pointer has a unique ID
- The index of a pointer can change from one event to the next, but the pointer ID of a pointer is guaranteed to remain constant as long as the pointer remains active.

MotionEvents events have action codes you access using getAction() or getActionMasked()

- ACTION_DOWN—For the first pointer that touches the screen. This starts the gesture. The pointer data for this pointer is always at index 0 in the MotionEvent.
- ACTION_POINTER_DOWN—For extra pointers that enter the screen beyond the first. The pointer data for this pointer is at the index returned by getActionIndex().
- ACTION_MOVE—A change has happened during a press gesture.
- ACTION_POINTER_UP—Sent when a non-primary pointer goes up.
- ACTION_UP—Sent when the last pointer leaves the screen.

A gesture is simply a series of touch events as follows:

1. DOWN. Begins with a single DOWN event when the user touches the screen
2. MOVE. Zero or more MOVE events when the user moves the finger around
3. UP. Ends with a single UP (or CANCEL) event when the user releases the screen

Scaling

For scaling, Android provides the ScaleGestureDetector class. ScaleGestureDetector uses the On ScaleGestureListener interface to implement all events or the SimpleScaleGestureListener interface to implement only some of the events. The best way to do this is to create our own custom class to handle scaling.

There are other gestures such as drag and drop, shake, and swipe that we're not going to cover today but all have similar structures.

Gestures

Create a new Android Studio project called gestures
Phone and Tablet min SDK API 19 (please chose 21 or lower or I won't be able to test your app)
Empty Activity
Activity Name: MainActivity
Layout Name: activity_main

Detecting All Common Gestures

Implement the GestureDetector.OnGestureListener and GestureDetector.OnDoubleTapListener interfaces.

```
public class MainActivity extends AppCompatActivity implements
GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener
```

You'll get red squiggles, click on them and then in the light bulb menu click implement methods. This will add all the required methods for these interfaces. Return true from a method indicates if you have handled the touch event, otherwise return false so the default action is processed for that event.

To understand when these are called we'll just add log statements in each method.

```
@Override
public boolean onSingleTapConfirmed(MotionEvent e) {
    Log.d("Gesture", "on single tap");
    return false;
}

@Override
public boolean onDoubleTap(MotionEvent e) {
    Log.d("Gesture", "on double tap");
    return false;
}

@Override
public boolean onDoubleTapEvent(MotionEvent e) {
    Log.d("Gesture", "on double tap event");
    return false;
}

@Override
public boolean onDown(MotionEvent e) {
    Log.d("Gesture", "on down");
    return false;
}

@Override
public void onShowPress(MotionEvent e) {
    Log.d("Gesture", "on show press");
}

@Override
public boolean onSingleTapUp(MotionEvent e) {
    Log.d("Gesture", "on single tap up");
    return false;
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
    Log.d("Gesture", "on scroll");
    return false;
}

@Override
public void onLongPress(MotionEvent e) {
    Log.d("Gesture", "on long press");
}

@Override
```

```
   public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
      Log.d("Gesture", "on fling");
      return false;
   }
```

Define an instance of the GestureDetector class.

```
private GestureDetector mGestureDetector;
```

Instantiate the GestureDetector instance.

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

   mGestureDetector = new GestureDetector(this, this);
}
```

Override the Activity's onTouchEvent() method, and pass along all observed events to the GestureDetector instance.

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

When you run it on a device open up the Logcat window and filter on Debug. Try different gestures and you'll see the log messages.
In a real app you would use these methods for your custom implementation for any of the gestures.

<u>Detecting a Subset of Common Gestures</u>
(Gestures pinch)
If you only want to override some gestures, you don't need to implement both interfaces.
Remove the interface implementation.

```
public class MainActivity extends AppCompatActivity
```

You'll have errors now so comment out all the methods that had been added EXCEPT the onTouchEvent() method.

Create a custom class

```
class CustomGestureDetector extends GestureDetector.SimpleOnGestureListener{
}
```

Implement the methods for any gestures that you want to process. For double tap implement onDoubleTapEvent(MotionEvent e).
You MUST override the onDown() method and return true since all gestures begin with onDown(). If you don't implement it false will be returned and the rest of the gesture will be ignored.

```
@Override
public boolean onDown(MotionEvent e) {
    Log.d("Gesture", "on down");
    return true;
}

@Override
public boolean onDoubleTapEvent(MotionEvent e) {
    Log.d("Gesture", "on double tap event");
    return false;
}
```

Update onCreate() so the gesture detector uses this new class.

```
mGestureDetector = new GestureDetector(this, new CustomGestureDetector());
```

Now if you run it you will only see messages for onDown every time you tap and for double taps.

Single View

Now let's remove the TextView and add an ImageView and only implement the custom gestures on the ImageView.

In the layout file remove the TextView and add an ImageView. I used an image of BB-8 and made the id of the ImageView bb8. I used a fixed size(112 x 112 dp) for the image so it was large enough to easily tap but didn't take up the majority of the view.

I commented out onTouchEvent() since I don't want to assign my gesture detector for all the views in the class.

Create a class level variable for our ImageView.

```
private ImageView bb8;
```

In onCreate() I got access to my ImageView and set the onTouchListener to the ImageView.

```
mGestureDetector = new GestureDetector(this, new CustomGestureDetector());

bb8 = (ImageView) findViewById(R.id.bb8);

final View.OnTouchListener onTouchListener = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        mGestureDetector.onTouchEvent(event);
        return true;
    }
};

bb8.setOnTouchListener(onTouchListener);
```

In onDoubleTapEvent() I added a toast message so I didn't have to keep looking at Logcat.

```
    Toast toast = Toast.makeText(getApplicationContext(), "I'm BB-8", Toast.LENGTH_SHORT);
    toast.show();
```

Now when you run it you should no longer see the logcat messages with the Gesture tag when you tap anywhere in the view, only on the image. You will see the standard onTouchEvent() calls which trigger an ACTION_MOVE event whenever the current touch contact position, pressure, or size changes.

Scaling

Now let's implement the pinch gesture using the ScaleGestureListener.
First, add a ScaleGestureDetector instance.

```
private ScaleGestureDetector mScaleDetector;
```

Create a new class for the scale gesture and implement 3 of its methods.

```
class CustomScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener{
private float scale = 1f;

   @Override
   public boolean onScale(ScaleGestureDetector detector) {
      scale = scale * detector.getScaleFactor();
      bb8.setScaleX(scale);
      bb8.setScaleY(scale);
      return true;
   }

   @Override
   public boolean onScaleBegin(ScaleGestureDetector detector) {
      Log.d("Gesture", "on scale begin");
      return true;
   }

   @Override
   public void onScaleEnd(ScaleGestureDetector detector) {
      Log.d("Gesture", "on scale end");
      super.onScaleEnd(detector);
   }
}
```

update onCreate() to create the scale detector.

```
mScaleDetector = new ScaleGestureDetector(this, new CustomScaleListener());
```

Add it to the OnTouchListener for the image view.

```
final View.OnTouchListener onTouchListener = new View.OnTouchListener() {
   @Override
   public boolean onTouch(View v, MotionEvent event) {
      mGestureDetector.onTouchEvent(event);
      mScaleDetector.onTouchEvent(event);
      return true;
   }
};
```

Run the app and use two fingers for the pinch gesture.

<u>Motion</u>
(Gestures image)
To handle the touch event and the drag gesture add the following at the class level.

```
// The active pointer is the one currently moving our object.
private int mActivePointerId = MotionEvent.INVALID_POINTER_ID;
// last x coordinate location
private float lastTouchX;
// last x coordinate location
private float lastTouchY;
```

Then update onCreate() to handle the different motion events.

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

   bb8 = (ImageView) findViewById(R.id.bb8);

   //gesture detector
   mGestureDetector = new GestureDetector(this, new CustomGestureDetector());

   //scale detector
   mScaleDetector = new ScaleGestureDetector(this, new CustomScaleListener());

   //touch for drag
   final View.OnTouchListener onTouchListener = new View.OnTouchListener() {
      @Override
      public boolean onTouch(View v, MotionEvent event) {

         mGestureDetector.onTouchEvent(event);
         mScaleDetector.onTouchEvent(event);

         //returns action code
         final int action = event.getActionMasked();

         Log.d("Gesture", "onTouch " + action);

         switch (action) {
            case MotionEvent.ACTION_DOWN: { //0
           //get pointer index
               final int pointerIndex = event.getActionIndex();
               final float x = event.getRawX();
               final float y = event.getRawY();

               lastTouchX = x;
               lastTouchY = y;

               mActivePointerId = event.getPointerId(pointerIndex);
               break;
            }
            case MotionEvent.ACTION_MOVE: { //2
               final float x = event.getRawX();
               final float y = event.getRawY();

               //calculate distance moved
               final float dx = x - lastTouchX;
               final float dy = y - lastTouchY;

               //set x and y position
               v.setX(x - 193);
               v.setY(y - 400);

               v.invalidate();

               lastTouchX = x;
               lastTouchY = y;

               break;
            }
            case MotionEvent.ACTION_UP: { //1
               mActivePointerId = MotionEvent.INVALID_POINTER_ID;
               break;
            }
            case MotionEvent.ACTION_CANCEL: {
               mActivePointerId = MotionEvent.INVALID_POINTER_ID;
               break;
            }
            case MotionEvent.ACTION_POINTER_UP: { //6 handles multiple pointers

               final int pointerIndex = event.getActionIndex();
               final int pointerId = event.getPointerId(pointerIndex);

               if (pointerId == mActivePointerId) {
                   final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
                   lastTouchX = event.getX(newPointerIndex);
                   lastTouchY = event.getY(newPointerIndex);
                   mActivePointerId = event.getPointerId(newPointerIndex);
               }
               break;
            }
         }
         return true;
   };
   bb8.setOnTouchListener(onTouchListener);
}
```

Now when you run it you should also be able to drag the image around the screen.