AND240

Touch in Android

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Handle single-touch events
2. Capture multi-touch interactions

Handle single-touch events

Xamarin University

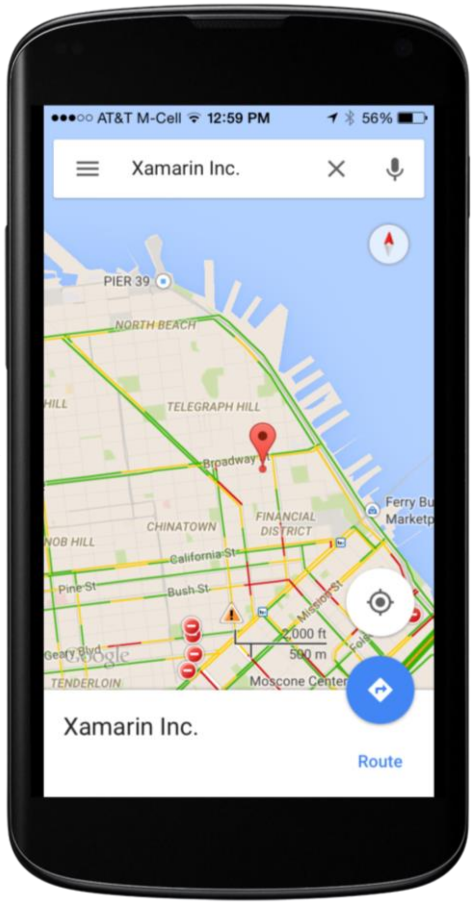
Tasks

1. Motivate touch UI
2. Determine event type
3. Subscribe to touch events
4. Utilize event data

Touch UI benefits

- Touch allows *direct* interaction with content

Touch is intuitive, it mimics how users work with physical objects →



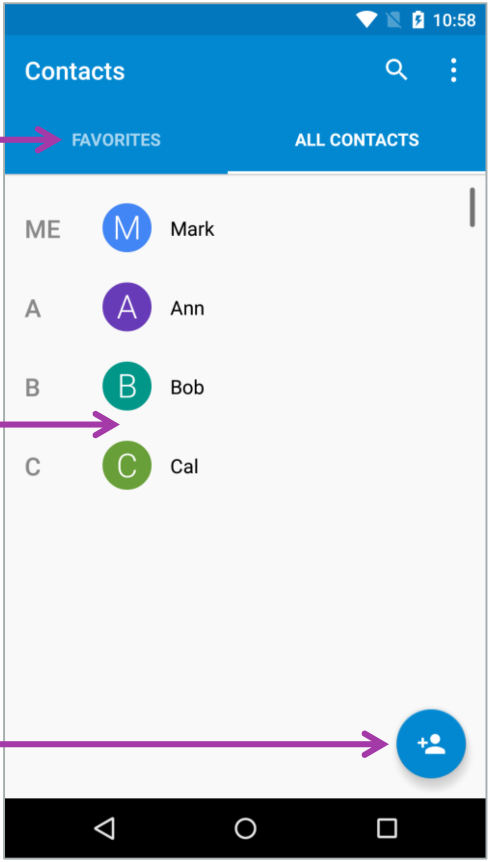
Challenges

- Designing for touch can be challenging; it requires careful attention to the placement and size of touchable items

Difficult to reach when used with one hand

8dp minimum spacing

48x48dp minimum size



Google layout guidelines: google.com/design/spec/layout/metrics-keylines.html

Touch vs. mouse design

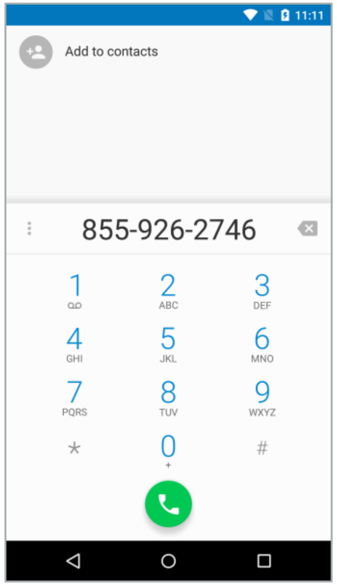
- Design considerations differ between Touch and Mouse; the patterns that work for desktop UI will not succeed in a touch-based UI

Touch	Mouse
Low precision for targets	High precision for targets
No right-click	Right-click available
No cursor	Cursor visible
Fingers will obscure the screen	Mouse will not obscure the screen
Portrait and landscape views	Only one view
Multiple fingers	One pointer

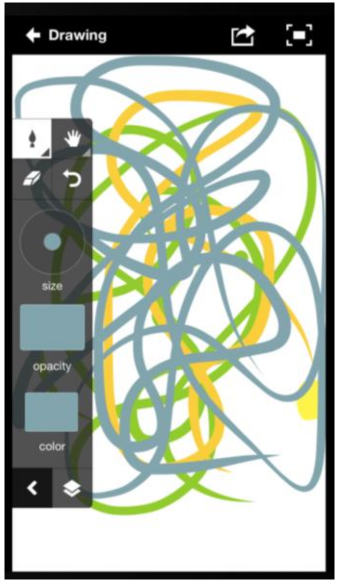
User events

❖ Android sends you both high-level events (e.g. button-click) and low-level events (e.g. touch)

Button and text-changed events are sufficient to build this UI →



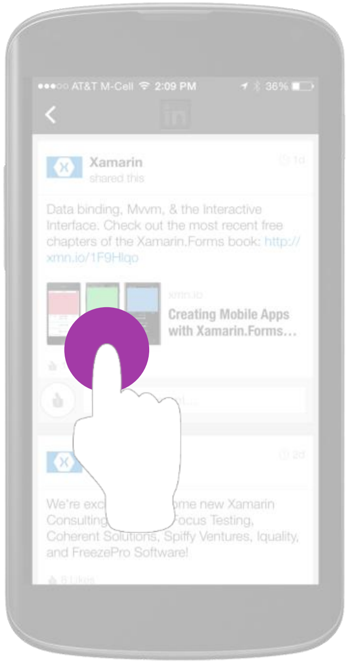
Drawing, Photo, and Games need low-level touch events →



The remainder of this course discusses the physical touch events

What is a touch action?

- An *action* is a physical manipulation of the screen e.g. down/move/up



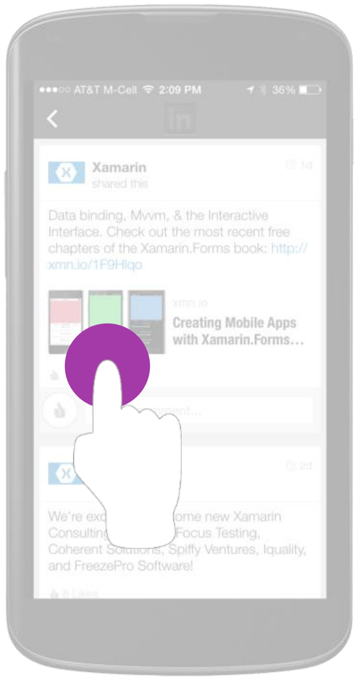
Make initial contact

Move an existing touch point

Lift a finger from the screen

Action sequence

❖ A screen interaction is made up of a sequence of low-level touch actions



AT&T M-Cell 2:09 PM 36%
Xamarin
shared this
Data binding, Mvvm, & the Interactive Interface. Check out the most recent free chapters of the Xamarin.Forms book: http://xmn.io/1F9Hlqo
Creating Mobile Apps with Xamarin.Forms...
We're exc... some new Xamarin Consultin... Focus Testing, Coherent Solutions, Spiffy Ventures, Iquality, and FreezePro Software!

Reported action: `Down`

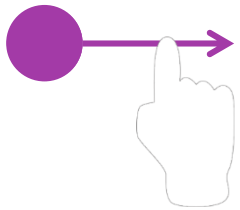
Reported action: `Move Move Move ...`

Reported action: `Up`

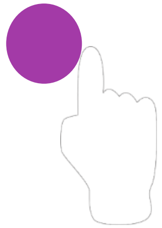
Actions

- `MotionEventActions` is an **`enum`** that reports the type of touch action

`MotionEventActions.Down`
Finger contacts the screen



`MotionEventActions.Move`
Finger moves on the screen



`MotionEventActions.Up`
Finger is lifted from the screen

Action reporting

- There is a single notification for all actions rather than separate down, move, and up notifications

```
Down
Move Move Move ...
Up
```

All actions reported through the same notification

Touch reporting

- The **`View`** class offers three ways to receive notifications of touch actions

```
public class View : ...
{ ...
  event EventHandler<View.TouchEventArgs> Touch;

  void SetOnTouchListener(View.IOnTouchListener l);

  virtual bool OnTouchEvent(MotionEvent e);
}
```

C#-style event → `event EventHandler<View.TouchEventArgs> Touch;`

Java-style listener → `void SetOnTouchListener(View.IOnTouchListener l);`

Override in a derived class → `virtual bool OnTouchEvent(MotionEvent e);`

The choice between the event and listener is personal preference.
The **`OnTouchEvent`** method is used when you code a derived class of **`View`**

Xamarin University

Touch data

- Touch notifications provide a **`MotionEvent`** containing the event details

All three reporting styles include a `MotionEvent` object

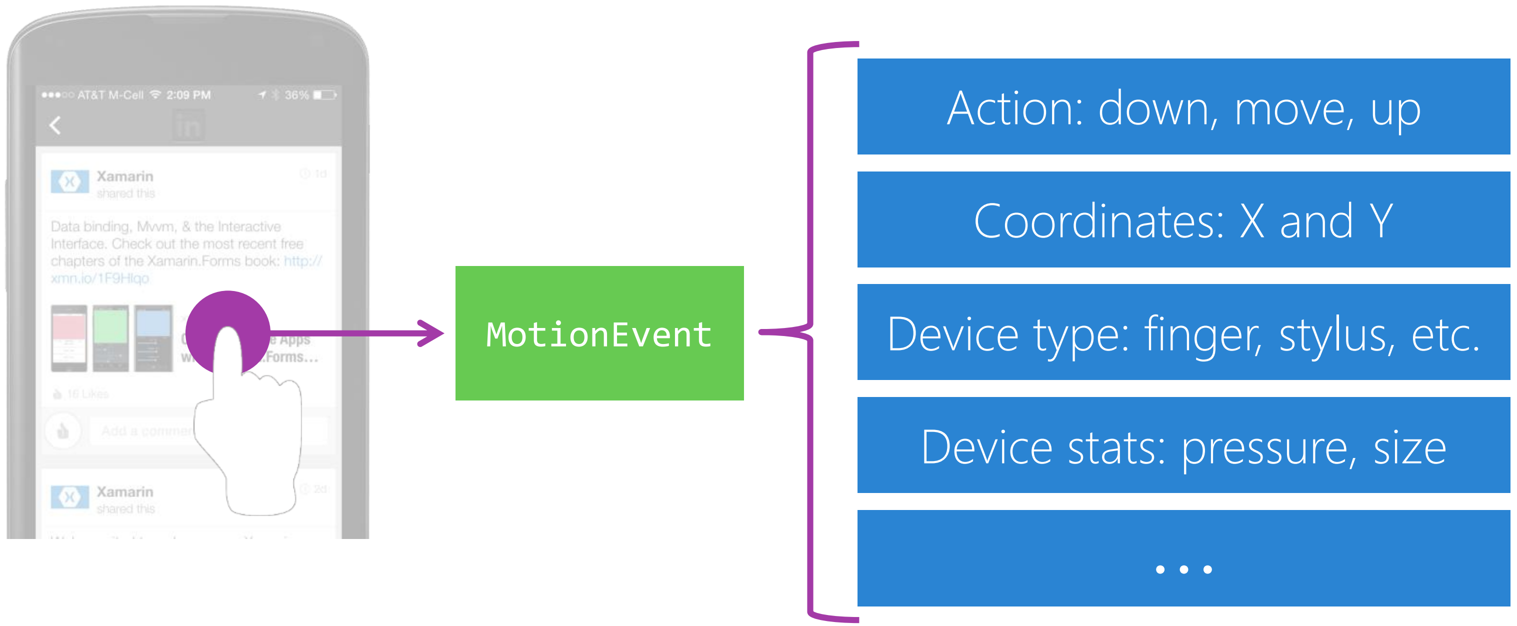
```
public class View : ...
{ ...
   event EventHandler<View.TouchEventArgs> Touch;

   void SetOnTouchListener(View.IOnTouchListener l);

   virtual bool OnTouchEvent(MotionEvent e);
}
```

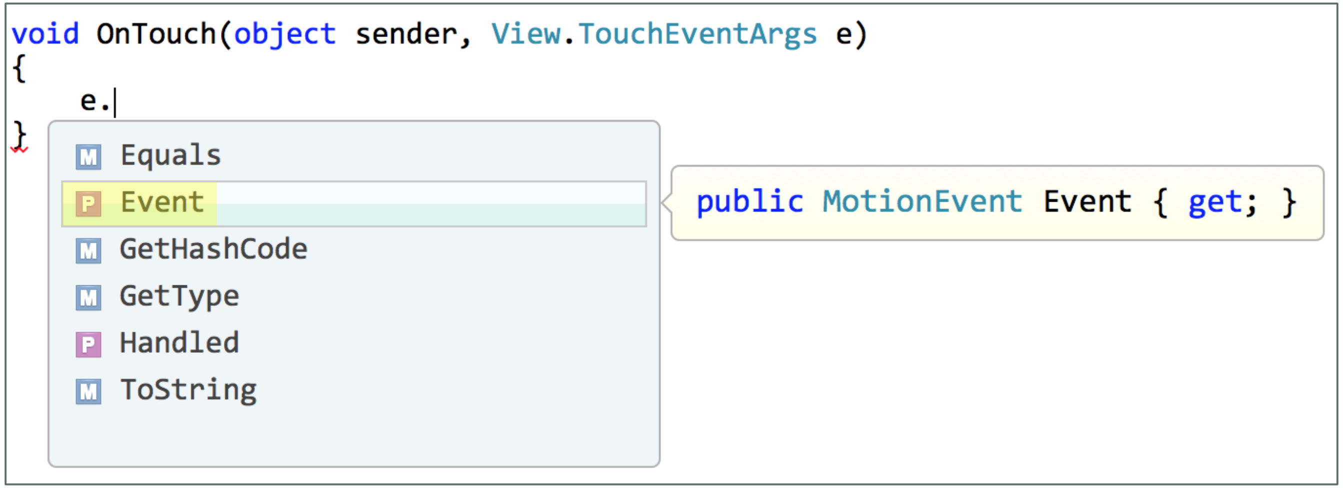
MotionEvent object

- `MotionEvent` includes information about the touch interaction


MotionEvent
Action: down, move, up
Coordinates: X and Y
Device type: finger, stylus, etc.
Device stats: pressure, size
...

Event property

❖ **`TouchEventArgs.Event`** provides the **`MotionEvent`** object for the event-style notification



```
void OnTouch(object sender, View.TouchEventArgs e)
{
    e.|
}
```

```
public MotionEvent Event { get; }
```

Xamarin University

How to determine the action

❖ The action is available in the **`MotionEvent.ActionMasked`** property

Determine the user action

```
void OnTouch(object sender, View.TouchEventArgs e)
{
  switch (e.Event.ActionMasked)
  {
    case MotionEventActions.Down: ... break;
    case MotionEventActions.Move: ... break;
    case MotionEventActions.Up  : ... break;
  }
}
```

You can also use the **`Action`** property to determine the user action; however, it contains multiple values so you must apply a bitmask: **`(e.Event.Action & MotionEventActions.Mask)`**

How to get the position

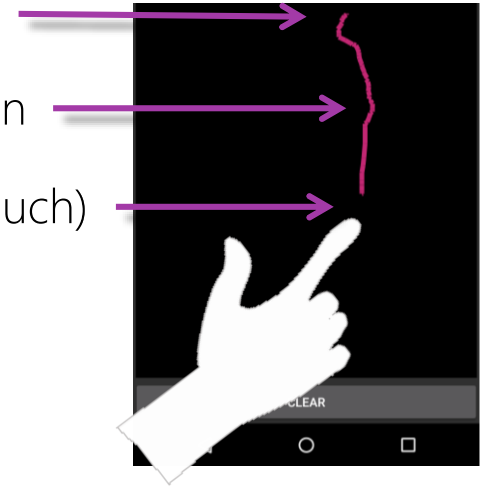
❖ The touch-event position is reported as X and Y coordinates relative to the view they are reported from, where `(0,0)` is the top-left

```
void OnTouch(object sender, View.TouchEventArgs e)
{
   ...
   float x = e.Event.GetX();
   float y = e.Event.GetY();
   ...
}
```

Example: Draw a line [concept]

❖ Touch events can be used to build a drawing app; for example, you could let the user draw a line with their finger

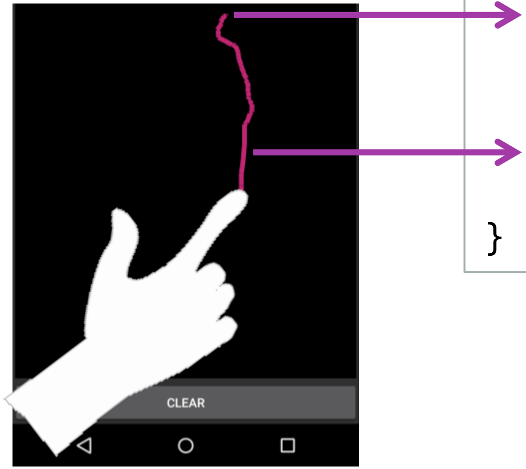
1. Begin the line on the **Down** action
2. Add a segment on the **Move** action
3. Reset on **Up** (not used in single-touch)



Xamarin University

Example: Draw a line [Path]

❖ The **`Path`** class represents a collection of geometric lines and curves



```
public class Path : Java.Lang.Object
{
  // begin a new segment (does not do any drawing)
  public virtual void MoveTo(float x, float y);

  // draw a line from previous point to this point
  public virtual void LineTo(float x, float y);
  ...
}
```

Example: Draw a line [implementation]

- You add points to the line as the user moves their finger

Only need one **`Path`**, it holds multiple lines →

```
Path p = new Path();
```

Get the coordinates →

Start a new line →

Add point to current line →

```
void OnTouch(object sender, View.TouchEventArgs e)
{
  var x = e.Event.GetX();
  var y = e.Event.GetY();

  switch (e.Event.ActionMasked)
  {
    case MotionEventActions.Down: p.MoveTo(x, y); break;
    case MotionEventActions.Move: p.LineTo(x, y); break;
  }
}
```

Individual Exercise

Create a drawing app using single-touch

Summary

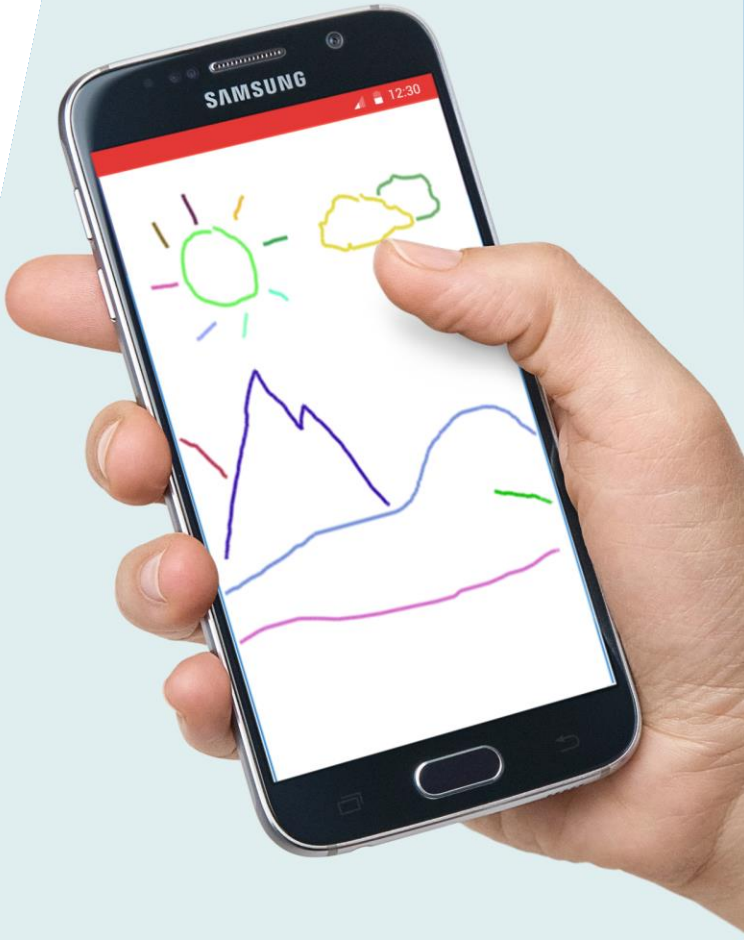
1. Motivate touch UI
2. Determine event type
3. Subscribe to touch events
4. Utilize event data

Capture multi-touch interactions

Xamarin University

Tasks

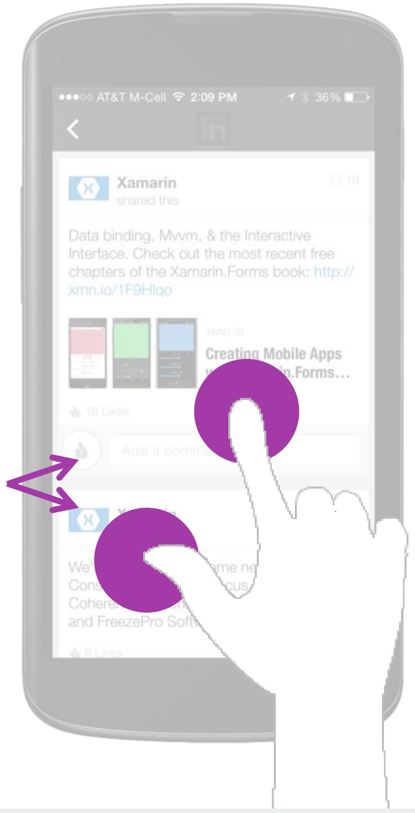
1. Discuss multi-touch in Android
2. Describe pointer motion event actions
3. Respond to multi-touch events
4. Create a multi-touch application



Multi-touch

❖ Most Android devices support multiple simultaneous touch points

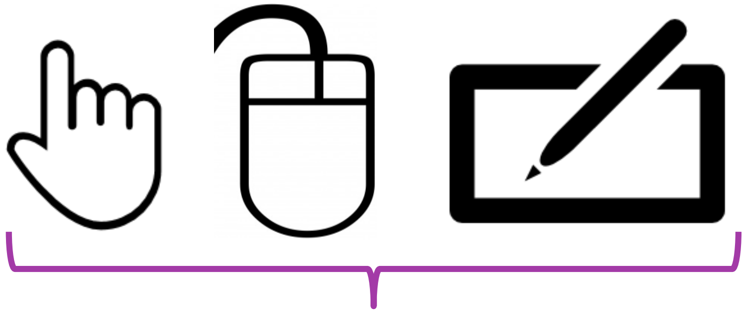
Down/Move/Up reported for both contact points



Can have popup "end cap" notes to point out final significant details

What is a pointer?

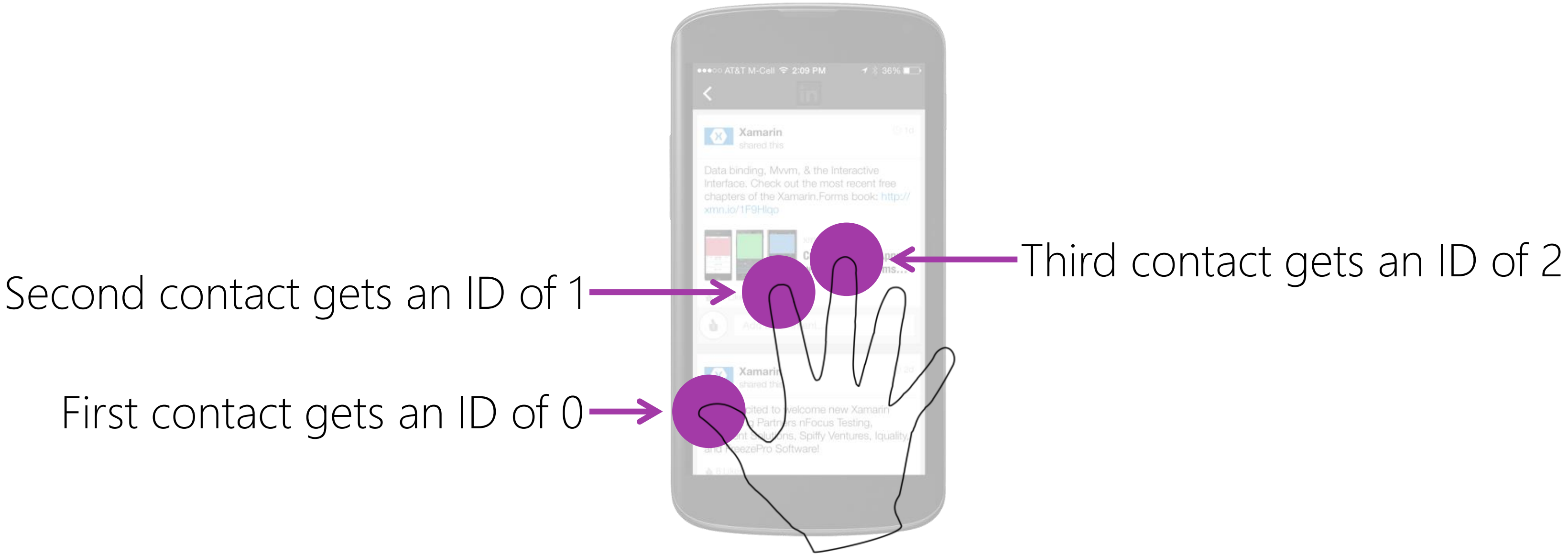
- *Pointer* is the generic term for any instrument that can interact with screen elements



Finger, mouse, and stylus are pointers

What is a pointer ID?

❖ A *pointer ID* is an integer that is assigned to a pointer when it first makes contact and remains the same until that pointer leaves the screen


Third contact gets an ID of 2
Second contact gets an ID of 1
First contact gets an ID of 0

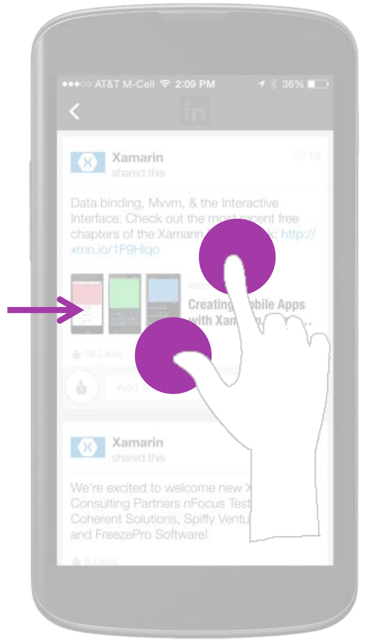
IDs are typically consecutive values starting at 0 but this is not guaranteed

Xamarin University

Primary vs. non-primary touch events

❖ Android differentiates between primary (first/only touch) and non-primary (subsequent) contacts

First-down and last-up are reported differently than other contacts



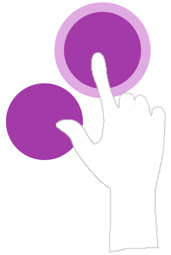
Android originally supported only single-touch; the multi-touch APIs were added later

Motion event actions for multi-touch

- Non-primary contacts are reported using specific values in the `MotionEventActions` enum

`MotionEventActions.PointerDown`

A new finger contacts the screen while there is already at least one other contact



`MotionEventActions.PointerUp`

A finger is lifted from the screen while there is still at least one other contact remaining

Multi-touch MotionEventActions

- There are 5 cases to handle for multi-touch down/move/up actions

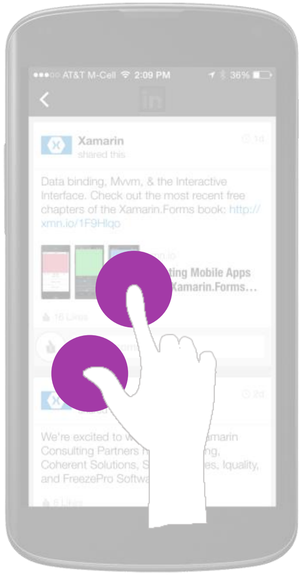
```
             void OnTouch (object sender, View.TouchEventArgs args)
             {
               switch (args.Event.ActionMasked)
               {
First-down  →    case MotionEventActions.Down:        ...
Other down  →    case MotionEventActions.PointerDown: ...

                 case MotionEventActions.Move:        ...

Other up    →    case MotionEventActions.PointerUp:   ...
Last-up     →    case MotionEventActions.Up:          ...
               }
             }
```

Multi-touch batched moves

- Down and up are reported as each pointer contacts or leaves the screen; however, move events report all current pointers at once



Sequence of reported actions:

```
Down
PointerDown
Move (includes new coords for both pointers)
Move (includes new coords for both pointers)
...
PointerUp
Up
```

Note that the position of some pointers may not change between move events

Xamarin University

Getting the number of active touches

- Android reports the number of active contacts in the `MotionEvent`'s `PointerCount` property

```
void OnTouch (object sender, View.TouchEventArgs args)
{
    int currentTouches = args.Event.PointerCount;
     ...
}
```

Number of current contacts

Group Exercise

MotionEventActions for multi-touch

Multi-touch data

- Android tracks data such as coordinates, stylus pressure, stylus size, etc. for all current touch points

Pointer ID: 0	Pointer ID: 1	Pointer ID: 2
X	X	X
Y	Y	Y
Pressure	Pressure	Pressure
...	...	...

There is one record for each active contact

Data storage

❖ Android does not say how data for the current contacts is stored; it can help to visualize it as an array with indices from `0` to `PointerCount-1`

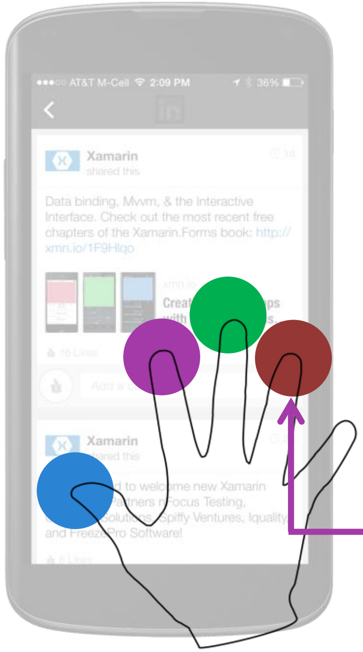
Index 0	Index 1	Index 2

`Pointer ID: 0`
`X`
`Y`
`Pressure`
`...`
	`Pointer ID: 1`
	`X`
	`Y`
	`Pressure`
	`...`
		`Pointer ID: 2`
		`X`
		`Y`
		`Pressure`
		`...`

`PointerCount` is 3 so indices are `0`,`1`,`2`

Adding records

❖ Typically, records for new contacts are added at the end (Note: Android says the exact ordering is not guaranteed)



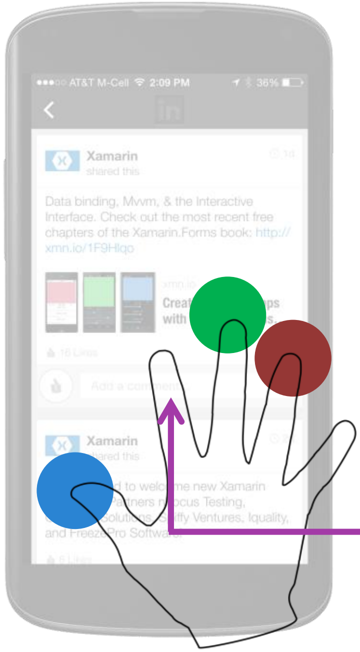
Index 0	Index 1	Index 2	Index 3
Pointer ID: 0	Pointer ID: 1	Pointer ID: 2	Pointer ID: 3
X	X	X	X
Y	Y	Y	Y
Pressure	Pressure	Pressure	Pressure
...	...	...	...

New pointer contacts the screen

New record added

Removing records

- As contacts leave the screen, their records are removed and the other records are compacted (this means Index and ID can get out of sync)



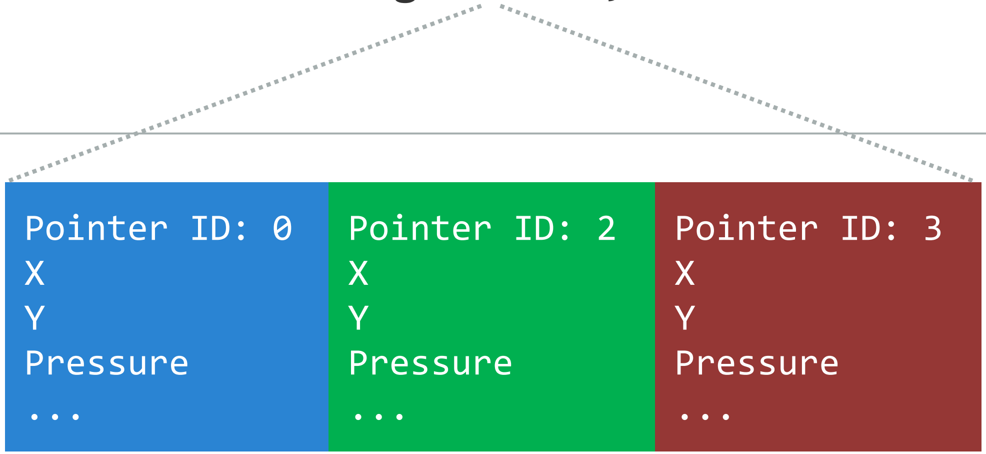
Index 0	Index 1	Index 2
Pointer ID: 0	Pointer ID: 2	Pointer ID: 3
X	X	X
Y	Y	Y
Pressure	Pressure	Pressure
...	...	...

Pointer leaves, record removed

MotionEvent data

❖ `MotionEvent` contains data for all current contacts

```
void OnTouch (object sender, View.TouchEventArgs args)
{
  MotionEvent e = args.Event;
  ...
}
```



Your event handler has access to data for all contacts

Accessing data

- `MotionEvent` has **`Get`** methods that take an index

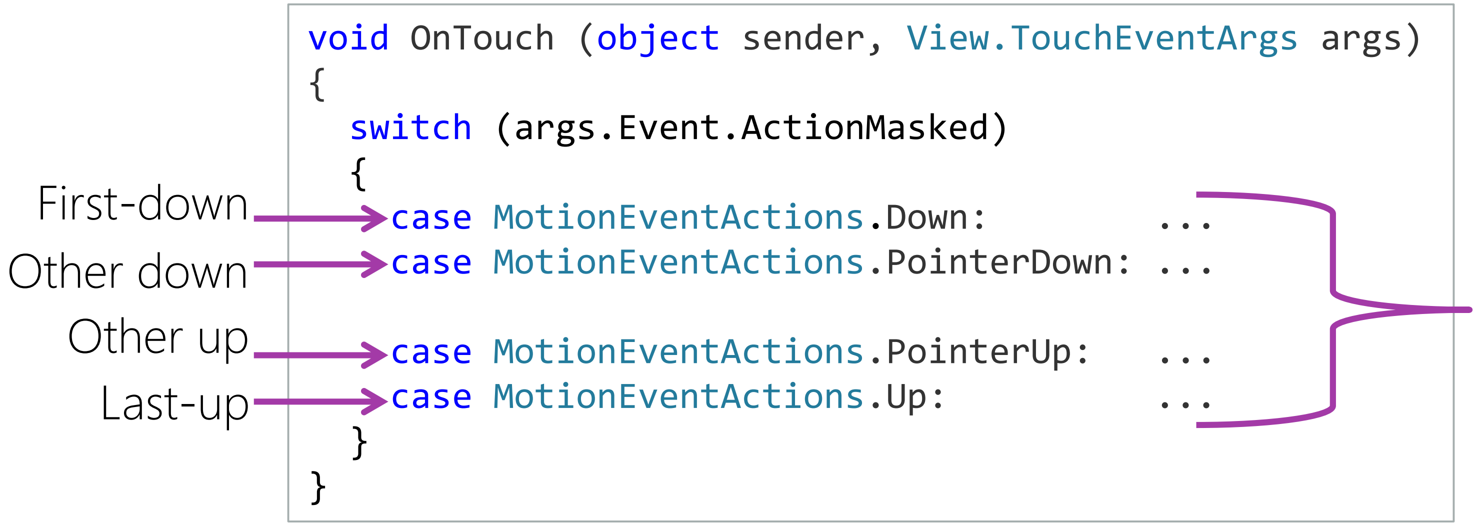
```
void OnTouch (object sender, View.TouchEventArgs args)
{
  for (int index = 0; index < args.Event.PointerCount; index++)
  {
    int   id = args.Event.GetPointerId(index);
    float x  = args.Event.GetX(index);
    float y  = args.Event.GetY(index);
  }
}
```

Index 0	Index 1	Index 2
Pointer ID: 0	Pointer ID: 2	Pointer ID: 3
X	X	X
Y	Y	Y
Pressure	Pressure	Pressure
...	...	...

Do not use the Pointer ID for access, it might not be a valid index

ActionIndex [motivation]

- For the down and up events, how do you know which element to look at to get the pointer ID, X, Y, etc.?



```
void OnTouch (object sender, View.TouchEventArgs args)
{
  switch (args.Event.ActionMasked)
  {
    case MotionEventActions.Down:        ...
    case MotionEventActions.PointerDown: ...

    case MotionEventActions.PointerUp:   ...
    case MotionEventActions.Up:          ...
  }
}
```

Where in the collection is the data in these cases?

Xamarin University

ActionIndex [Down and Up]

❖ **Down** and **Up** do not need any additional info; they report first-finger-down and last-finger-up so there is a single element in the collection

First-down

Last-up

```
void OnTouch (object sender, View.TouchEventArgs args)
{
   switch (args.Event.ActionMasked)
   {
     case MotionEventActions.Down: int id = args.Event.GetPointerId(0); ...
     case MotionEventActions.Up:   int id = args.Event.GetPointerId(0); ...
     ...
   }
}
```

Pass **0** to the **Get** methods

ActionIndex [definition]

❖ `ActionIndex` indicates the index of the non-primary pointer that contacted or left the screen during **`PointerDown`** and **`PointerUp`**

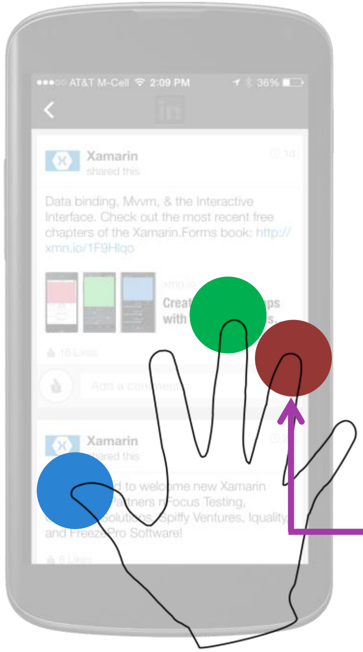
```
void OnTouch (object sender, View.TouchEventArgs args)
{
  switch (args.Event.ActionMasked)
  {
    case MotionEventActions.PointerDown: int id = args.Event.GetPointerId(args.Event.ActionIndex); ...
    case MotionEventActions.PointerUp:   int id = args.Event.GetPointerId(args.Event.ActionIndex); ...
    ...
  }
}
```

`ActionIndex` is only valid for these two actions

Access the element corresponding to the finger that caused this action

ActionIndex [PointerDown]

- **`PointerDown`** reports that a non-primary finger touched the screen; **`ActionIndex`** tells where the new record was added



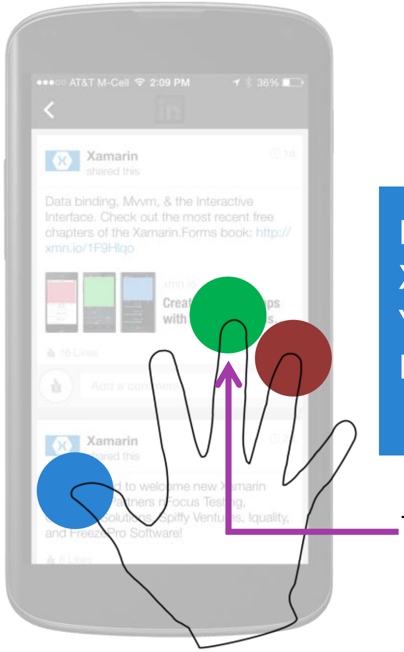
Index 0	Index 1	Index 2
`Pointer ID: 0`	`Pointer ID: 2`	`Pointer ID: 3`
`X`	`X`	`X`
`Y`	`Y`	`Y`
`Pressure`	`Pressure`	`Pressure`
`...`	`...`	`...`

New pointer contacts the screen

New record added here, **`ActionIndex`** will be 2

ActionIndex [PointerUp]

- `PointerUp` reports that a non-primary finger left the screen; `ActionIndex` tells which record is about to be deleted



Index 0	Index 1	Index 2
`Pointer ID: 0`	`Pointer ID: 2`	`Pointer ID: 3`
`X`	`X`	`X`
`Y`	`Y`	`Y`
`Pressure`	`Pressure`	`Pressure`
`...`	`...`	`...`

This Pointer leaves the screen

This record will be removed, `ActionIndex` is 1

Flash Quiz

Xamarin University

Flash Quiz

① The **`ActionIndex`** is best described as:

a) The pointer's position on the screen

b) The pointer's ID

c) The pointer's position in the collection

Flash Quiz

① The **`ActionIndex`** is best described as

a) The pointer's position on the screen

b) The pointer's ID

c) <u>The pointer's position in the collection</u>

Flash Quiz

② The **`ActionIndex`** is valid for which **`MotionEventActions`**?

a) `Down`

b) `PointerDown`

c) `Move`

d) `PointerUp`

e) `Up`

Xamarin University

Flash Quiz

② The **`ActionIndex`** is valid for which **`MotionEventActions`**?

a) `Down`

b) <u>`PointerDown`</u>

c) `Move`

d) <u>`PointerUp`</u>

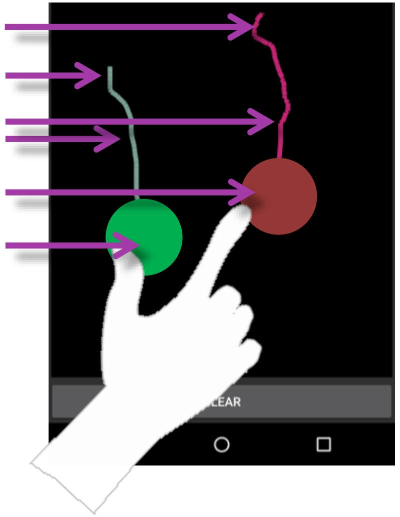
e) `Up`

Xamarin University

Example: Draw multiple lines [concept]

❖ Multi-touch events can be used to let the user drawer multiple lines at the same time

1. Begin a line on the **`Down`** action
2. Begin a line on the **`PointerDown`** action
3. Add points to all lines on the **`Move`** action
4. Stop adding points on **`PointerUp`** action
5. Stop adding points on the **`Up`** action



Xamarin University

Example: Draw multiple lines [begin]

- Create a new **Path** as each finger contacts the screen; store them in a dictionary keyed by pointer ID

All active lines →

```
Dictionary<int, Path> paths = new Dictionary<int, Path>();
```

Use **0** with **Get** methods on **Down** →

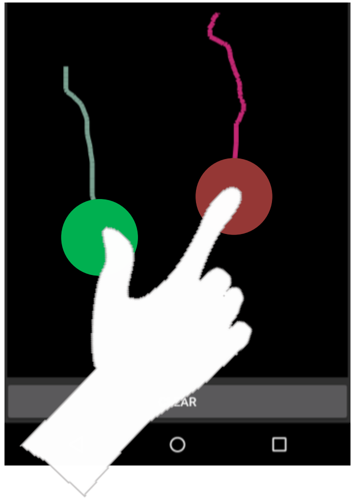
Use **ActionIndex** with **Get** methods on **PointerDown** →

```
case MotionEventActions.Down
  var path = new Path();
  paths.Add(e.Event.GetPointerId(0), path);
  path.MoveTo(e.Event.GetX(0), e.Event.GetY(0));
  break;

case MotionEventActions.PointerDown:
  var path = new Path();
  paths.Add(e.Event.GetPointerId(e.Event.ActionIndex), path);
  path.MoveTo(e.Event.GetX(e.Event.ActionIndex),
              e.Event.GetY(e.Event.ActionIndex));
  break;
```

Example: Draw multiple lines [move]

- On **Move**, use pointer ID to find the right **Path** object in the dictionary



Index 0	Index 1
Pointer ID: 0	Pointer ID: 2
X 200	X 50
Y 175	Y 225
Pressure	Pressure
...	...

You loop through this list

```
for (int i = 0; i < e.PointerCount; i++)
{
   int   id = e.Event.GetPointerId(i);
   float x  = e.Event.GetX(i);
   float y  = e.Event.GetY(i);

   var path = paths[id];

   path.LineTo(x, y);
}
```

Use the pointer ID to locate the correct line for this X/Y pair

Xamarin University

Example: Draw multiple lines [end]

- Remove a **Path** from the dictionary when its pointer leaves the screen

Use **ActionIndex** with **Get** method on **PointerUp** →

Use **0** with **Get** method on **Up** →

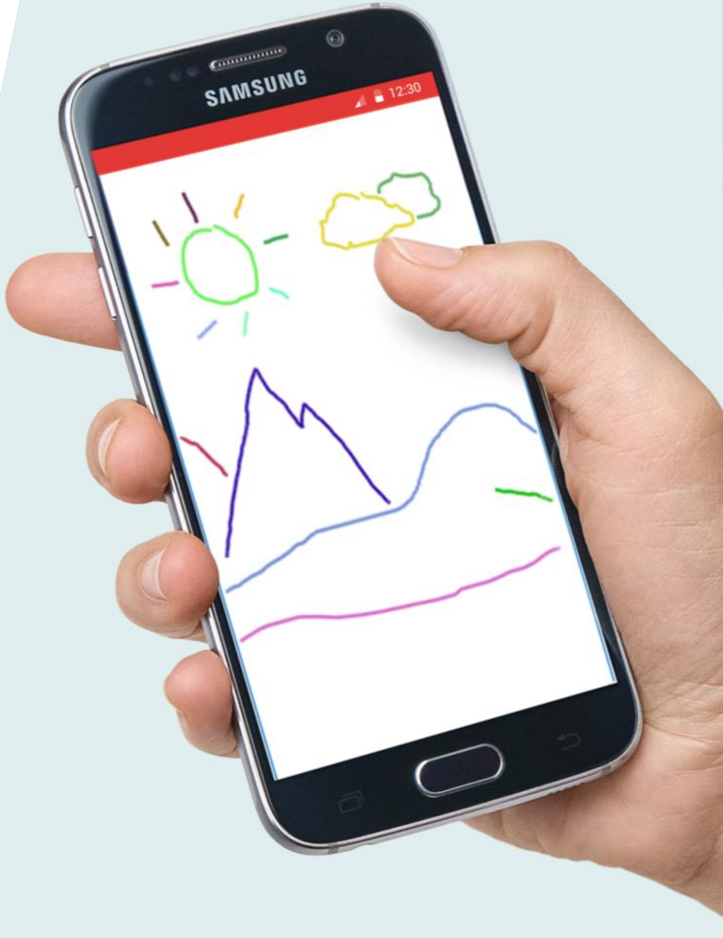
```
case MotionEventActions.PointerUp:
{
   int id = e.Event.GetPointerId(e.ActionIndex);
   paths.Remove(id);
   break;
}
case MotionEventActions.Up:
{
   int id = e.Event.GetPointerId(0);
   paths.Remove(id);
   break;
}
```

Individual Exercise

Create a multi-touch drawing app

Summary

1. Discuss multi-touch in Android
2. Describe pointer motion event actions
3. Respond to multi-touch events
4. Create a multi-touch application



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft